Thesis for the Degree of Doctor of Philosophy

Topics in Distributed Algorithms: On Wireless Networks, Distributed Storage and Streaming

Thomas Petig

Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden 2017

Topics in Distributed Algorithms:
On Wireless Networks, Distributed Storage and Streaming
Thomas Petig
ISBN 978-91-7597-673-0

Technical report 151D
Doktorsavhandlingar vid Chalmers tekniska högskola
Ny series nr 4354
ISSN 0346-718X

Department of Computer Science and Engineering
Chalmers University of Technology
412 96 Göteborg, Sweden
Phone: +46 (0)31-772 10 53

Author e-mail: petig@chalmers.se

Printed by Chalmers Reproservice
Göteborg, Sweden 2017

Topics in Distributed Algorithms: On Wireless Networks, Distributed Storage and Streaming

Thomas Petig
Chalmers University of Technology

ABSTRACT

Distributed algorithms are executed on a set of computational instances. We refer to these instances as nodes. Nodes are running concurrently and are independent from each other. Furthermore, they have their own instructions and information. In this context, the challenges are to show that the algorithm is correct, regardless of computational, or communication delays and to show bounds on the usage of communication. We are especially interested the behaviour after transient faults and under the existence of Byzantine nodes.

This thesis discusses fundamental communication models for distributed algorithms. These models are implementing abstract communication methods. First, we address medium access control for a wireless medium with guarantees on the communication delay. We discuss time division multiple access (TDMA) protocols for ad-hoc networks and we introduce an algorithm that creates a TDMA schedule without using external references for localisation, or time. We justify our algorithm by experimental results.

The second topic is the emulation of shared memory on message passing networks. Both, shared memory and message passing are basic interprocessor communication models for distributed algorithms. We are providing a way of emulating shared memory on top of an existing message passing network under the presence of data corruption and stop-failed nodes. Additionally, we ensure the privacy of the data that is stored in the shared memory.

The third topic looks into streaming algorithms and optimisation. We study the problem of sorting a stream ofvehicles on a highway with several lanes so that each vehicle reaches its target lane. We look into optimality in terms of minimising the number of move operations, as well as, minimising the length of the output stream. We present an exact algorithm for the case of two lanes and show that NP-Hardness for a increasing number of lanes.

Keywords: Distributed Algorithm, Time Division Multiple Access, TDMA, Wireless Networks, Shared Memory, Message Passing, Fault-Tolerance, Streaming, Optimisation, NP-Hardness

Acknowledgements

I express my gratitude to all my current and former colleagues at the Department of Computer Science and Engineering for providing this research environment, especially: Hiva Alahyari, Magnus Almgren, Martina Brachmann, Olaf Landsiedel, Herbert Lange, Aljoscha Lautenbach, Ioannis Nikolakopoulos, Elena Pagnin and Daniel Schöpe. I thank especially my friends and colleagues Iosif Salem and Valentin Tudor for five nice years in our office EDIT 5126, as well as my flat mate Aras Atalar[1]. This thesis would not be possible without the help of the administration of the CSE department.

I thank the people who I have met and from whom I have learned a lot, this includes: Henning Brandt von Lindau, Sebastian Freitag, Melanie Haars, Marian Risse, Alexej Smoljanov, Viviane Zwanger. Cristina Caprio, Andrea Fabio Michael Martinangeli Simona and Valentin Tudor became even better friends after an excellent visit to Romania. Furthermore, I appreciate a lot to got the opportunity to spend all these hours on the water with Ole Martin Christensen, Jan Gustav Grolig, Steffen Hammer, Jessica Köster, Viktor Nilsson and Tuule Soniste and the members of ChSS. I am especially glad for comments and discussions with Tanja Zerenner. I thank my opponent, Volker Turau, for his very valuable comments on this thesis.

Last, but not least, I thank my Family, Reinhard, Ilse, Eckart and Christine for their support.

Thomas Petig

[1] With special guest Paul Renaud Goud.

List of Appended Papers

First Paper *"Self-stabilising TDMA Algorithms for Wireless Ad-hoc Networks without External Reference"*, Thomas Petig, Elad M. Schiller, Philippas Tsigas. Extended version of what appeared as brief announcement in the proceedings of 15th International Symposium Stabilization, Safety, and Security of Distributed Systems (SSS), November 13-16, 2013, Osaka, Japan and as extended abstract in the proceedings of 13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET) 2014, Piran, Slovenia and as technical report, CoRR, abs/1308.6475, 2013.

Second Paper *"DecTDMA: A Decentralized-TDMA with Link Quality Estimation for WSNs"*, Olaf Landsiedel, Thomas Petig and Elad M. Schiller. Appeared as extended abstract in the proceedings of Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium (SSS) 2016, Lyon, France.

Third Paper *"Robust and Private Distributed Shared Atomic Memory in Message Passing Networks"*, Shlomi Dolev, Thomas Petig, Elad M. Schiller. Based on the brief announcement as it appears in the proceedings of ACM Symposium on Principles of Distributed Computing (PODC) 2015, Donostia-San Sebastián, Spain.

Fourth Paper *"Changing Lanes on a Highway"*, Thomas Petig, Elad M. Schiller and Jukka Suomela

List of Figures

Contents

1 Introduction

This work studies topics in distributed and centralised algorithms. In distributed algorithms, we consider models with many separate nodes that store only limited information locally. We assume that the nodes are connected and can exchange messages via communication channels [9]. The nodes execute a given algorithm concurrently and independently. But, they do not necessarily execute the same instructions. In distributed algorithms, we look into problems as complexity and fault tolerance. Complexity can, for example, be measured in how much time does it take, or how many messages need to be transmitted, to archive a common goal, or how much data needs to be stored locally. In the context of fault tolerance, we discuss the options for recovering after a fault or being resilient to faults. We address the recovery by using self-stabilisation while resilience against data corruption is handled by an error correction algorithm. A further aspect we present, is privacy in distributed storage algorithm. Here we ensure that data that is stored on a set of nodes can not be revealed by a small set of nodes, where small means the cardinality is less than a given constant.

Distributed algorithms are used within distributed systems, e.g., computer networks for decentralised storage and computation. These systems are the backbone of many modern technologies, e.g., cloud computing, and there size is increasing. Therefore, we need efficient algorithms that keep the requirements for hardware and communication channels low. Furthermore, the required level of autonomy is increasing with the increasing size of such systems. Faults should be handled, if possible, without human interaction to keep these systems maintainable and to ensure the system availability is high. Note that real hardware is prone to failure. Thus, the probability that all nodes in a system are working is decreasing exponentially with the increasing number of nodes, even without taking the reliability of the network into account. This motivates us for studying the complexity and fault tolerance of distributed algorithms.

Within the area of centralised algorithms, we look into optimisation algorithms and NP-hardness results. Although, we discuss an centralised approach, the results provide bounds on the complexity of a distributed solution. In fact, our studied problem looks into move operations of a set of agents.

In the following, we see a which models are used in the presented papers and discuss similarities and differences between them. We continue by a look into fault-tolerance. We close the introduction with a summary of the presented papers and possible extensions.

1.1 Models

The discussion of distributed algorithms is dependent on the communication model that is used to describe the architecture. The presented papers are using different models, but

with some similarities. Note that the second paper provides an example of an implementation of the first one and does not discuss theoretical aspects. In the first and the third paper, we consider systems that consist of a set of nodes $\mathcal{P}$ that can execute a program. These nodes have a certain communication functionality, so nodes can exchange messages with each other. In both publications we are using a communication graph $G := (\mathcal{P}, E)$. The nodes are the vertices in this graph and every pair of nodes that can communicate are represented by an edge in this graph, i.e., $E \subset \mathcal{P} \times \mathcal{P}$.

One of the differences in the models is the way how nodes can exchange information across the edges of the communication graph. In the first paper, a node can communicate with all neighbours in the communication graph by using a broadcast. Such a broadcast submits a single packet to all neighbours in the communication graph at once. This models the behaviour of a wireless transmission. We assign to each broadcast a time interval to model the transmission time in the wireless medium. Based on the overlap of those intervals we allow an adversary to drop packets due to a collision. Other possible ways—that embed the nodes in a Euclidean space—are for example the unit disc graph model [6], or the SINR model [2].

In the third paper, we assume an asynchronous setting. This means the execution at different nodes run with different speeds. Furthermore, we assume that the delay on the communication channel is unbounded. The communication is based on message passing networks. Here the node can send individual messages to each neighbour. These messages are always delivered, i.e., message loss is not possible. But, the uncertainty comes from the unbounded communication time. The nodes are divided into three sets, writers, servers, and readers. The edges are the union of all pairs of servers with each other, all pairs of servers with writers and all pairs of servers with readers. The analysis within the underlying model is borrowed from [3].

The fourth paper does not discuss a communication model, but the movement of agents. We place agents on a two dimensional grid and allow movement along edges to neighbouring grid locations. A constraint, that we require, allows at most one agent per location. This setup is close to the "15" Puzzle [7]. We call one dimension of the grid rows and the other one lanes. In contrast to the puzzle, we only require each agent to be on a given target lane, but we do not require a specific row in the solution. The optimization goal is to minimise the number of move operations until every agent reaches is target lane.

1.2 Fault-Tolerance

We consider two kinds of faults—transient faults and data corruption. To address transient faults we utilise self-stabilisation, as it was introduced in 1974 by Edsger W. Dijkstra with his work "Self-stabilising Systems in Spite of Distributed Control" [4]. For limiting the impact of data corruption we are looking into erasure codes [13] and error correction algorithms.

To show self-stabilisation we assume a given communication graph $G := (\mathcal{P}, E)$. The *state*, s_{p_i}, of a node $p_i \in \mathcal{P}$ consists of the value of all the variables of the node in-

cluding the set of all incoming communication channels. The execution of an algorithm step can change the state. The term *configuration* is used for a tuple of the form $(s_{p_1}, s_{p_2}, \ldots, s_{p_n})$, where each s_{p_i} is the state of node p_i. We define an *execution* $R = (c(0), a(0), c(1), a(1), \ldots)$ as an alternating sequence of system configurations $c(x)$ and steps $a(x)$, such that each configuration $c(x+1)$, except the initial configuration $c(0)$, is obtained from the preceding configuration $c(x)$ by the execution of the step $a(x)$. For a finite execution $R' := (c'(0), a'(0), c'(1), \ldots, c'(k))$ and an execution $R'' := (c''(0), a''(0), c''(1), \ldots)$, such that $c'(k) = c''(0)$, we define $\circ$ as the operator that maps R' and R'' to the concatenated execution $R' \circ R'' := (c'(0), a'(0), \ldots, c'(k), a''(0), c''(1), \ldots)$. We call R' prefix and R'' suffix. For $R := (c(0), a(0), \ldots, c(k))$, we say that R's length is $k-1$, because it includes $k-1$ steps. A task $\mathcal{T}$ is a set of constraints for a configuration that we require the system to fulfil. We define the set of legal executions as the set of executions LE that fulfil $\mathcal{T}$ at every configuration. A safe configuration denotes a configuration, such that every execution that starts from this configuration is included in LE. Every execution R of an self-stabilising algorithm can then be written as $R = R' \circ R''$ where R'' is a legal execution. The convergence time is then a bound on R'.

The data corruption is our model introduced by semi-Byzantine nodes. These nodes do not manipulate the communication protocol itself, but they can send corrupted embedded data. To address data corruption we are using a special case of maximum distance separable (MDS) codes, so called Reed-Solomon codes [13]. These codes are in the class of the well-known erasure codes. Erasure codes map an input vector v to an output vector of higher dimension, w, such that this map is invertible. The mapping ensures that the inverse map is still defined and mapping to the same v if several entries of w are erased. This property is commonly used, e.g., on optical storage mediums, where scratches can make parts of the medium unreadable, but it is still possible to read the original data using these erasure codes. By using Reed-Solomon codes we archive resilience against these semi-Byzantine attacks. For this we use the error correction of Berlekamp and Welch [16]. This allows the reader to reconstruct the data even if up to a constant amount of server deliver corrupted coded elements.

1.3 Summary

In this thesis some fundamental communication models for distributed algorithms are discussed. These models provide certain guarantees for distributed applications, like fault tolerance, privacy and bounded communication delay.

1.3.1 First Paper: Self-stabilising TDMA Algorithms for Wireless Ad-hoc Networks without External Reference

We address time division multiple access (TDMA) protocols for decentralised ad-hoc networks. TDMA protocols provide, in contrast to carrier sense multiple access (CSMA), higher throughput, and bounded communication delays if the radio channel is available. This is useful to provide a medium access control (MAC) to an application with guarantees on the message delivery time. To solve this problem, we have to solve a chicken-egg

problem: after a transient fault one cannot rely on the MAC, but one need the communication between nodes to reorganise the MAC.

We look into a model where every node has a local clock, but we do not assume access to external references for localisation, or time. The task $\mathcal{T}$ is to synchronise all local clocks and to agree on a partition of time into intervals of equal size, called frames. Every frame is partitioned into a constant number of equal sized time slots. Additionally, the task includes the unique assignment of nodes to time slots, such that no two nodes with an edge distance of two, or less, share the same interval. We assume that this assignment is the same for every configuration in a legal execution.

We provide a lower bound for the number of time slots per frame for the task $\mathcal{T}$, i.e., if locally too many nodes are competing for a time slot, they might block each other. Furthermore, we propose a self-stabilising algorithm and by this it can recover from transit faults. This algorithm is, to our knowledge, the first algorithm with these properties. To synchronise the local clocks we utilise a converge-to-the-max approach as it was introduced by Hermann and Zhang [5]. One important point in communication graphs with a diameter larger than 1 is the resolution of two-hop conflicts [12]. In such a conflict a node $p \in \mathcal{P}$ receives messages from two different neighbours concurrently. And, therefore, it might not be able to receive any of them, due to the collision. We address this topic by randomly utilising unassigned time slot for transmitting additional messages and using these messages the proposed algorithm can break symmetry.

1.3.2 Second Paper: DecTDMA: A Decentralized-TDMA with Link Quality Estimation for WSNs

After discussing theoretical results in the first paper, we look into experimental results. We provide an example of an implementation of the Algorithm of the first paper on TelosB motes. Then we compare with an implementation, that is named DecTDMA, tailored to cope with unreliable links.

One of the draw backs of the self-stabilising algorithm presented in the first paper is that packet loss is treated as transient fault. This prevents the system from stabilisation in case there are unreliable links. The self-stabilising algorithm requires every message to be acknowledged by all neighbours, otherwise it assumes a message loss due to conflicting time slots. In practice, links that successfully deliver messages with probability 1 do not exist. Especially in wireless sensor networks the success rate degenerates with distance.

The provided DecTDMA algorithm is not self-stabilising. It estimates for each neighbour the link quality, i.e., the probability of successful messages delivery to that neighbour. Based on this it computes an expect number of acknowledgements and compares the measured value with the expected value. A backoff is initiated if the measured value does not meet the expectation. On the down side, this approach is not able to handle systematic message loss do to conflicting time slots. But, the practical results are significantly better than the self-stabilising algorithm.

1.3.3 Third Paper: Robust and Private Distributed Shared Atomic Memory in Message Passing Networks

Shared memory and message passing are essential communication models for distributed algorithms. Therefore, simulation shared memory on a message passing architecture is an important problem. The literature in this area includes the for us relevant publications of Attiya et al. [1] for single-writer multi-reader (SWMR) and for multi-writer multi-reader (MWMR) by Lynch and Shvartsman [10] [1].

We present an algorithm for emulation shared memory on message passing networks. We focus on a multi-writer multi-reader shared memory, where we have a set of writers that can write to a common shared memory and a set of readers that can read from it. Concurrent write and read operations are explicitly allowed. The shared memory is emulated by a set of servers. Note that we do not consider self-stabilisation. Our work extends the work of Cadambe et al. on coded shared atomic memory algorithm for message passing architectures [3]. They show a way to emulate shared memory on a message passing network using maximum distance separable codes, a subclass of erasure codes [13]. By using erasure codes, Cadambe et al. are ensuring that stop failed servers can be tolerated.

To handle semi-Byzantine attacks, our proposed algorithm uses Reed-Solomon codes with error correction. But, we show even more, because in this context, this leads to additional properties. It was shown by McEliece and Sarwate in 1981 [11] that Reed-Solomon codes can be used to implement Shamir's secret sharing [14]. Using this we add privacy, i.e., one has to steal at least k coded elements to reconstruct the data, where k is a parameter of the used code.

1.3.4 Fourth Paper: Changing Lanes on a Highway

This work studies the problem of sorting agents between streams under the constraint that each agent requires space for navigation. One could see this as vehicles on a highway with several lanes that would like to reach there target lane. We look into optimality for the case of two lanes in terms of minimising the number of move operations, as well as, minimising the length of the output stream. We present an exact algorithm for the case of two lanes and show that NP-Hardness for a increasing number of lanes. We allow to change between neighbouring lanes and neighbouring rows if the destination is free. This setting is similar to combinatorial puzzles [17], such as the "15" Puzzle [7].

[1] These are of course not the first works on shared memory emulation, earlier works are for example [15] and [8].

Bibliography

[1] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. "Sharing memory robustly in message-passing systems". In: *J. ACM (JACM)* 42.1 (1995), pp. 124–142.

[2] Chen Avin, Yuval Emek, Erez Kantor, Zvi Lotker, David Peleg, and Liam Roditty. "SINR diagrams: towards algorithmically usable SINR models of wireless networks". In: *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*. Ed. by Srikanta Tirthapura and Lorenzo Alvisi. ACM, 2009, pp. 200–209. ISBN: 978-1-60558-396-9. DOI: 10.1145/1582716.1582750. URL: http://doi.acm.org/10.1145/1582716.1582750.

[3] Viveck R. Cadambe, Nancy A. Lynch, Muriel Médard, and Peter M. Musial. "A Coded Shared Atomic Memory Algorithm for Message Passing Architectures". In: *2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014, Cambridge, MA, USA, 21-23 August, 2014*. IEEE Computer Society, 2014, pp. 253–260. ISBN: 978-1-4799-5392-9. DOI: 10.1109/NCA.2014.44. URL: http://dx.doi.org/10.1109/NCA.2014.44.

[4] Edsger W. Dijkstra. "Self-stabilizing Systems in Spite of Distributed Control". In: *Commun. ACM* 17.11 (1974), pp. 643–644. DOI: 10.1145/361179.361202. URL: http://doi.acm.org/10.1145/361179.361202.

[5] Ted Herman and Chen Zhang. "Best Paper: Stabilizing Clock Synchronization for Wireless Sensor Networks". In: *SSS*. Ed. by Ajoy Kumar Datta and Maria Gradinariu. Vol. 4280. LNCS. Springer, 2006, pp. 335–349. ISBN: 978-3-540-49018-0.

[6] Mark L Huson and Arunabha Sen. "Broadcast scheduling algorithms for radio networks". In: *Military Communications Conference, 1995. MILCOM'95, Conference Record, IEEE*. Vol. 2. IEEE. 1995, pp. 647–651.

[7] Wm. Woolsey Johnson and William E. Story. "Notes on the "15" Puzzle". English. In: *American Journal of Mathematics* 2.4 (1879), pp. 397–404. ISSN: 00029327. URL: http://www.jstor.org/stable/2369492.

[8] Leslie Lamport. "Concurrent Reading and Writing". In: *Commun. ACM* 20.11 (1977), pp. 806–811. DOI: 10.1145/359863.359878. URL: http://doi.acm.org/10.1145/359863.359878.

[9] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. ISBN: 1-55860-348-4.

[10] Nancy A. Lynch and Alexander A. Shvartsman. "Robust Emulation of Shared Memory Using Dynamic Quorum-Acknowledged Broadcasts". In: *Digest of Papers: FTCS-27, The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing, Seattle, Washington, USA, June 24-27, 1997*. IEEE Computer Society, 1997, pp. 272–281. ISBN: 0-8186-7831-3. DOI: 10.1109/FTCS.1997.614100. URL: http://dx.doi.org/10.1109/FTCS.1997.614100.

[11] R. J. McEliece and D. V. Sarwate. "On Sharing Secrets and Reed-Solomon Codes". In: *Commun. ACM* 24.9 (1981), pp. 583–584. ISSN: 0001-0782. DOI: 10.1145/358746.358762. URL: http://doi.acm.org/10.1145/358746.358762.

[12] Stéphane Pomportes, Joanna Tomasik, Anthony Busson, and Véronique Vèque. "Self-stabilizing Algorithm of Two-Hop Conflict Resolution". In: *12th Inter. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*. 2010, pp. 288–302.

[13] Ron M. Roth. *Introduction to coding theory*. Cambridge Press, 2006. ISBN: 978-0-521-84504-5.

[14] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[15] Paul M. B. Vitányi and Baruch Awerbuch. "Atomic Shared Register Access by Asynchronous Hardware (Detailed Abstract)". In: *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 233–243. ISBN: 0-8186-0740-8. DOI: 10.1109/SFCS.1986.11. URL: http://dx.doi.org/10.1109/SFCS.1986.11.

[16] L.R. Welch and E.R. Berlekamp. *Error correction for algebraic block codes*. US Patent 4,633,470. 1986. URL: https://www.google.com/patents/US4633470.

[17] Richard M Wilson. "Graph puzzles, homotopy, and the alternating group". In: *Journal of Combinatorial Theory, Series B* 16.1 (1974), pp. 86–96.

PAPER I

Thomas Petig, Elad M. Schiller, Philippas Tsigas

Self-stabilizing TDMA Algorithms for Wireless Ad-hoc Networks without External Reference

Appeared as technical report
CoRR, abs/1308.6475
http://arxiv.org/abs/1308.6475
2013

As extended abstract in the proceedings of
13th Annual Mediterranean Ad Hoc Networking Workshop (MED-HOC-NET)
Piran, Slovenia
June 2-4, 2014, pp. 87–94

As brief announcement in the proceedings of
15th International Symposium Stabilization, Safety, and Security of Distributed Systems
Osaka, Japan
November 13-16, 2013, pp. 354–356

2 Self-stabilizing TDMA Algorithms for Wireless Ad-hoc Networks without External Reference

Time division multiple access (TDMA) is a method for sharing communication media. In wireless communications, TDMA algorithms often divide the radio time into timeslots of uniform size, ξ, and then combine them into frames of uniform size, τ. We consider TDMA algorithms that allocate at least one timeslot in every frame to every node. Given a maximal node degree, δ, and no access to external references for collision detection, time or position, we consider the problem of collision-free probabilistic stabilising TDMA algorithms that use constant frame size.

We demonstrate that this problem has no solution when the frame size is τ is less than $\max\{2\delta, \chi_2\}$, where χ_2 is the chromatic number for distance-2 vertex colouring. As a complement to this lower bound, we focus on proving the existence of collision-free probabilistic stabilising TDMA algorithms that use constant frame size of τ. We consider basic settings (no hardware support for collision detection and no prior clock synchronization), and the collision of concurrent transmissions from transmitters that are at most two hops apart. In the context of probabilistic stabilising systems that have no external reference, we are the first to study this problem (to the best of our knowledge), and use simulations to show convergence even with computation time uncertainties.

2.1 Introduction

Autonomous and cooperative systems will ultimately carry out risk-related tasks, such as piloting driverless cars, and liberate mankind from mundane labour, such as factory and production work. Note that the implementation of these cooperative systems implies the use of wireless ad hoc networks and their critical component – the *medium access control* (MAC) layer. Since cooperative systems operate in the presence of people, their safety requirements include the provision of real-time guarantees, such as constant communication delay. Infrastructure-based wireless networks successfully provide high bandwidth utilization and constant communication delay. They divide the radio into *timeslots* of uniform size, ξ, that are then combined into *frames* of uniform size, τ. Base-stations, access points or wireless network coordinators can schedule the frame in a way that enables each node to transmit during its own timeslot, and arbitrate between nearby nodes that wish to communicate concurrently. We strive to provide the needed MAC protocol properties, using limited radio and clock settings, i.e., no external reference for collision detection, time or position. Note that ad hoc networks often do not consider collision detection mechanisms, and external references are subject to signal loss. For these settings, we demonstrate that there is no solution for the studied problem when the frame size is $\tau < \max\{2\delta, \chi_2\}$, where δ is a bound on the node degree, and χ_2 is the chromatic number for distance-2 vertex colouring. The main result is the existence of collision-free probabilistic stabilising TDMA algorithms that use constant frame size of $\tau > \max\{4\delta, X_2\}+1$, where $X_2 \geq \chi_2$ is a number that depends on the colouring algorithm in use. To the best of our knowledge, we are the first to study the problem of probabilistic stabilising TDMA timeslot allocation without external reference. The algorithm simulations demonstrate feasibility in a way that is close to the practical realm.

Wireless ad hoc networks have a dynamic nature that is difficult to predict. This gives rise to many fault-tolerance issues and requires efficient solutions. These networks are also subject to transient faults due to temporal malfunctions in hardware, software and other short-lived violations of the assumed system settings, such as changes to the communication graph topology. We focus on fault-tolerant systems that recover after the occurrence of transient faults, which can cause an arbitrary corruption of the system state (so long as the program's code is still intact). These *self-stabilising* [9] design criteria simplify the task of the application designer when dealing with low-level complications, and provide an essential level of abstraction. Consequently, the application design can easily focus on its task – and knowledge-driven aspects. Probabilistic stabilisation [13] ensures that the recovery happens with probability 1.

ALOHAnet protocols [1] are pioneering MAC algorithms that let each node select one timeslot per TDMA frame at random. In the Pure Aloha protocol, nodes may transmit at any point in time, whereas in the Slotted Aloha version, the transmissions start at the timeslot beginning. The latter protocol has a shorter period during which packets may collide, because each transmission can collide only with transmissions that occur within its timeslot, rather than with two consecutive timeslots as in the Pure Aloha case. Note that the random access approach of ALOHAnet cannot provide constant communication delay. Distinguished nodes are often used when the application requires

bounded communication delays, e.g., IEEE 802.15.4 and deterministic self-stabilising TDMA [2, 15]. Without such external references, the TDMA algorithms have to align the timeslots while allocating them. Existing algorithms [4] circumvent this challenge by assuming that $\tau/(\Delta+1) \geq 2$, where Δ is an upper bound on the number of nodes with whom any node can communicate with using at most one intermediate node for relaying messages. This guarantees that every node can transmit during at least one timeslot, s, such that no other transmitter that is at most two hops away, also transmits during s. However, the $\tau/(\Delta+1) \geq 2$ assumption implies bandwidth utilization that is up to $\mathcal{O}(\delta)$ times lower than the proposed algorithm, because $\Delta \in \mathcal{O}(\delta^2)$.

As a basic result, we show that $\tau/\delta \geq 2$, and as a complement to this lower bound, we focus on considering the case of $\tau/\delta \geq 4$. We present a collision-free probabilistic stabilising TDMA algorithm that use constant frame size of τ. We show that it is sufficient to guarantee that collision freedom for a single timeslot, s, and a *single* receiver, rather than *all* neighbours. This narrow opportunity window allows control packet exchange, and timeslot alignment. After convergence, there are no collisions of any kind, and each frame includes at most one control packet.

Related work Herman and Zhang [12] assume constant bounds on the communication delay and present self-stabilising clock synchronization algorithms for wireless ad hoc networks. Herman and Tixeuil [11] assume access to synchronized clocks and present the first self-stabilising TDMA algorithm for wireless ad hoc networks. They use external reference for dividing the radio time into timeslots and assign them according to the neighbourhood topology. The self-stabilisation literature often does not answer the causality dilemma of "which came first, synchronization or communication" that resembles Aristotle's *'which came first, the chicken or the egg?'* dilemma. On one hand, existing clock synchronization algorithms often assume the existence of MAC algorithms that offer bounded communication delay, e.g. [12], but on the other hand, existing MAC algorithms that provide bounded communication delay, often assume access to synchronized clocks, e.g. [11]. We propose a bootstrapping solution to the causality dilemma of "which came first, synchronization or communication", and discover convergence criteria that depend on τ/δ.

The *converge-to-the-max synchronization* principle assumes that nodes periodically transmit their clock value, ownClock. Whenever they receive clock values, receivedClock $>$ ownClock, that are greater than their own, they adjust their clocks accordingly, i.e., ownClock $\leftarrow$ receivedClock. Herman and Zhang [12] assume constant bounds on the communication delay and demonstrate convergence. Basic radio settings do not include constant bounds on the communication delay. We show that the converge-to-the-max principle works when given bounds on the expected communication delay, rather than constant delay bounds, as in [12].

The proposal in [10] considers shared variable emulation. Several self-stabilising algorithms adopt this abstraction, e.g., a generalized version of the dining philosophers problem for wireless networks in [6], topology discovery in anonymous networks [20], random distance-k vertex colouring [21], deterministic distance-2 vertex colouring [3], two-hop conflict resolution [25], a transformation from central demon models to distributed scheduler ones [27], to name a few. The aforementioned algorithms assume that

if a node transmits infinitely many messages, all of its communication neighbours will receive infinitely many of them. We do not make such assumptions about *(underlying) transmission fairness*. We assume that packets, from transmitters that are at most two hops apart, can collide *every time*.

The authors of [16] present a MAC algorithm that uses convergence from a random starting state (inspired by self-stabilisation). In [17, 23], the authors use computer network simulators for evaluating self-$\star$ MAC algorithms. A self-stabilising TDMA algorithm, that accesses external time references, is presented in [18]. Simulations are used for evaluating the heuristics of MS-ALOHA [26] for dealing with timeslot exhaustion by adjusting the nodes' individual transmission signal strength. We provide an algorithm to solve this while considering basic radio settings. The results presented in [14, 7] do not consider the time it takes the algorithm to converge, as we do. We mention a number of MAC algorithms that consider onboard hardware support, such as receiver-side collision detection [7, 26, 5, 29, 4]. We consider merely basic radio technology that is commonly used in wireless ad hoc networks. The MAC algorithms in [29, 28] assumes the accessibility of an external time or geographical references or the node trajectories, e.g., Global Navigation Satellite System (GNSS). We instead integrate the TDMA timeslot alignment with clock synchronization.

Our contribution Given a maximal node degree, δ, we consider the problem of the existence of collision-free probabilistic stabilising TDMA algorithms that use constant frame size of τ. In the context of (probabilistic) self-stabilising systems that have no external reference, we are the first to study this problem (to the best of our knowledge). The proposed probabilistic stabilising and bootstrapping algorithm answers the causality dilemma of synchronization and communication.

For settings that have no assumptions about fairness and external reference existence, we establish a basic limit on the bandwidth utilization of TDMA algorithms in wireless ad hoc networks (Section 2.3). Namely, $\tau < \max\{2\delta, \chi_2\}$, where χ_2 is the chromatic number for distance-2 vertex colouring. We note that the result holds for general graphs with a clearer connection to bandwidth utilization for the cases of tree graphs ($\chi_2 = \delta+1$) and planar graphs [22] ($\chi_2 = 5\delta/3 + \mathcal{O}(1)$).

We claim the existence of collision-free probabilistic stabilising TDMA algorithms that use constant frame size of τ without assuming the availability of external references (Section 2.4). We also demonstrate convergence of the concept via simulations that take uncertainties into account, such as (local) computation time.

2.2 System Settings

The system consists of a set, $P := \{p_i\}_i$, of communicating entities, which we call *nodes*. An upper bound, $\nu > |P|$, on the number of nodes in the system is known. Subscript font is used to point out that X_i is p_i's variable (or constant) X. Node p_i has a unique identifier, id_i, that is known to p_i but not necessarily by $p_j \in P \setminus \{p_i\}$.

Communication graphs With $\delta_i \subseteq P$ we denote the set of nodes with which a node $p_i \in P$ can communicate. The system can be represented by an undirected network

of directly communicating nodes, $G := (P, E)$, named the *communication graph*, where $E := \{\{p_i, p_j\} \in P \times P : p_j \in \delta_i\}$. We assume that G is connected. For $p_i, p_j \in P$, we define the distance, $d(p_i, p_j)$, as the number of edges in an edge minimum path connecting p_i and p_j. We denote by $\Delta_i := \{p_j \in P : 0 < d(p_i, p_j) \leq 2\}$ the 2-neighbourhood of p_i, and the upper bounds on the sizes of δ_i and Δ_i are denoted by $\delta \geq \max_{p_i \in P}(|\delta_i|)$, and respectively, $\Delta \geq \max_{p_i \in P}(|\Delta_i|)$. We assume that $\mathsf{diam} \geq \max_{p_i, p_j \in P} d(p_i, p_j)$ is an upper bound on the network diameter.

Synchronization The nodes have fine-grained clock hardware (with arbitrary clock offset upon system start). For the sake of presentation simplicity, our work considers zero clock skews, i.e., the clocks of all nodes run at the same speed. We denote with clock tick, the time interval it takes for the clocks to increment by 1. We assume that the *clock* value, $C \in \{0, \ldots, \mathcal{C} - 1\}$, and any timestamp in the system have $\mathcal{C}$ states. The pseudo-code uses the GetClock() function that returns a timestamp of C's current value. We say that the clocks are *synchronized* when $\forall p_i, p_j \in P : C_i = C_j$, where C_i is p_i's clock value. Since the clock value can overflow at its maximum, and wrap to the zero value, arithmetic expressions that include timestamp values are module $\mathcal{C}$, e.g., the function AdvanceClock$(x) := C \leftarrow (C + x) \bmod \mathcal{C}$ adds x time units to clock value, C, modulo its number of states, $\mathcal{C}$.

Periodic pulses invoke the MAC protocol, and divide the radio time into *(broadcasting) timeslots* of ξ time units in a way that provides sufficient time for the transmission of a single packet. We group τ timeslots into *(broadcasting) frames*. The pseudo-code uses the event *timeslot*() that is triggered by the event $0 = C_i \bmod \xi$ and $s() := C_i \div \xi \bmod \tau$ is the *timeslot number*, where $\div$ is the integer division.

Operations The communication allows a message exchange between the sender and the receiver. After the sender, p_i, fetches message $m \leftarrow \text{MACfetch}_i()$ from the upper layer, and before the receiver, p_j, delivers it to the upper layer in $\text{MACdeliver}_j(m)$, they exchange m via the operations $transmit_i(m)$, and respectively, $m \leftarrow receive_j()$. We model the communication channel, $q_{i,j}$ (queue), from node p_i to node $p_j \in \delta_i$ as the most recent message that p_i has sent to p_j and that p_j is about to receive, i.e., $|q_{i,j}| \leq 1$. When p_i transmits message m, the operation $transmit_i(m)$ inserts a copy of m to every $q_{i,j}$, such that $p_j \in \delta_i$. Once m arrives, p_j executes *receive*() and returns the tuple $\langle i, t_i, t_j, m \rangle$, where $t_i = C_i$ and $t_j = C_j$ are the clock values of the associated $transmit_i(m)$, and respectively, $m \leftarrow receive_j()$ calls. We assume zero propagation delay and efficient time-stamping mechanisms for t_i and t_j. Moreover, the timeslot duration, ξ, allows the transmission and reception of at least a single packet, see Property 1.

Property 1. Let $p_i \in P$, $p_j \in \delta_i$. At any point in time t_i in which node p_i transmits message m for duration of ξ, node p_j receives m if there is no node $p_k \in (\delta_i \cup \delta_j) \setminus \{p_i\}$ that transmits starting from time t_k with duration ξ such that $[t_i, t_i + \xi)$ and $[t_k, t_k + \xi)$ are intersecting.

This means a node can receive a message if no node in the neighbourhood of the sender and no node in the neighbourhood of the receiver is transmitting concurrently.

Interferences Wireless communications are subject to interferences when two or more neighbouring nodes transmit *concurrently*, i.e., the packet transmission periods

overlap or intersect. We model communication interferences, such as unexpected peaks in ambient noise level and concurrent transmissions of neighbouring nodes, by letting the *(communication) environment* to selectively omit messages from the communication channels. We note that we do *not* consider any error (collision) indication from the environment.

The environment can use the operation $omission_{i,j}(m)$ for removing message m from the communication channel, $q_{i,j}$, when p_i's transmission of m to $p_j \in \delta_i$ is concurrent with the one of $p_k \in \Delta_i$. Immediately after $transmit_i(m)$, the environment selects a subset of p_i's neighbours, $Omit_m \subseteq \delta_i$, removes m from $q_{i,j} : p_j \in Omit_m$ and by that it prevents the execution of $m \leftarrow receive_j()$. Note that $Omit_m = \delta_i$ implies that no direct neighbour can receive message m.

Probabilistic stabilisation Every node, $p_i \in P$, executes a program that is a sequence of *(atomic) steps*, a_i. The state, st_i, of node $p_i \in P$ includes p_i's variables, including the clocks and the program control variables, and the communication channels, $q_{i,j} : p_j \in \delta_i$. The *(system) configuration* is a tuple $c := (st_1, \ldots, st_{|P|})$ of node states. Given a system configuration, c, we define the set of *applicable steps*, $a = \{a_i\}$, for which p_i's state, st_i, encodes a non-empty communication channel or an expired timer. An *execution* is an unbounded alternating sequence $R := (c(0), a(0), c(1), a(1), \ldots)$ (Run) of configurations $c(k)$, and applicable steps $a(k)$ that are taken by the algorithm and the environment. The task $\mathcal{T}$ is a set of specifications and LE (legal execution) is the set of all executions that satisfy $\mathcal{T}$. We say that configuration c is *safe*, when every execution that starts from it is in LE. An algorithm is called *probabilistic self-stabilising* [13, 8] if it reaches a safe configuration with probability 1.

Task definition We consider the task $\mathcal{T}_{\text{TDMA}}$, that requires all nodes, p_i, to have timeslots, s_i, that are uniquely allocated to p_i within Δ_i. We define LE_{TDMA} to be the set of legal executions, R, for which $\forall p_i \in P : (p_j \in P \Rightarrow C_i = C_j) \wedge (((s_i \in \{0, \ldots, \tau - 1\}) \wedge (p_j \in \Delta_i)) \Rightarrow s_i \neq s_j)$ holds in all of R's configurations. We note that for a given finite τ, there are communication graphs for which $\mathcal{T}_{\text{TDMA}}$ does not have a solution, e.g., the complete graph, $K_{\tau+1}$, with $\tau + 1$ nodes. In Section 2.3, we show that the task solution can depend on the (arbitrary) starting configuration, rather then just the communication graph.

2.3 Basic Results

We establish a basic limitation of the bandwidth utilization for TDMA algorithms in wireless ad hoc networks. Before generalizing the limitation, we present an illustrative example (Lemma 1) of a starting configuration for which $\tau < \max\{2\delta, \chi_2\}$, where χ_2 is the chromatic number for distance-2 vertex colouring.

Lemma 1. *Let $\delta \in \mathbb{N}$ and $\tau < 2\delta$. Suppose that the communication graph, $G := (\{p_0, \ldots p_\delta\}, E)$, has the topology of a star, where the node p_δ is the centre (root) node and $E := \{p_\delta\} \times L$, where $L := \{p_0, \ldots p_{\delta-1}\}$ are the leaf nodes. There is a starting configuration $c(x)$, such that an execution R starting from $c(x)$ of any algorithm solves the task $\mathcal{T}_{\text{TDMA}}$ does not converge.*

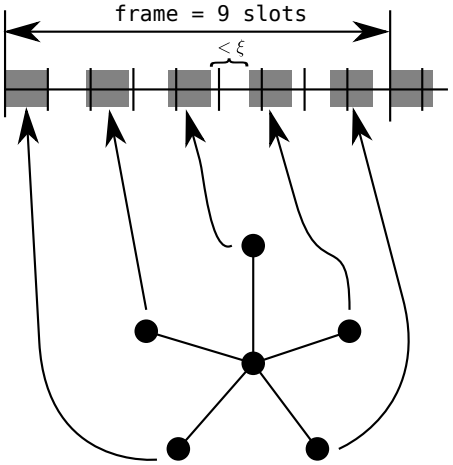


Figure 2.1: The outer five nodes are covering nine timeslots. The top horizontal line and its perpendicular marks depict the radio time division according to the central node, p_δ. The grey boxes depict the radio time covered by the leaf nodes, $p_i \in L$.

Proof. We prove this Lemma by showing an example in which we assign timeslots to a subset of nodes in a way, such that they block each other and, thus, disconnect the communication graph.

Let $\tau = 2\delta - 1$. Let $c(x)$ be such that: (1) C_i in $c(x)$ has the properties $(C_i + (2\xi - 1)i) \bmod \xi = 0$ and $(C_i + (2\xi - 1)i) \div \xi \bmod \tau = s_i$ for all $p_i \in L \setminus \{p_\delta\}$, (2) $s_\delta = \perp$ and (3) there is no message in transit. Figure 2.1 shows such a graph for $\delta = 5$. This means the next timeslot of node $p_i \in L \setminus \{p_\delta\}$ starts $(2\xi - 1)i$ clock ticks after $c(x)$. The gap between the time p_i's timeslot ends and p_{i+1}'s timeslot starts is $(2\xi-1)(i+1)-((2\xi-1)i+\xi) = \xi-1 < \xi$ clock ticks long and, thus, smaller than a time slot. The gap between the next transmission of $p_{\delta-1}$ and the next next transmission of p_0 is $(2\delta - 1)\xi + (2\xi - 1)0 - ((2\xi - 1)(\delta - 1) + \xi) = \delta - 1 < \xi$. This pattern repeats, because only p_δ receives these messages transmitted by the leaves and p_δ does not have a time slot assigned and according to Property 1 any attempt of p_δ in transmitting can fail. Thus, no algorithm can establish communication here. □

The proof of Lemma 1 considers that case of $\tau < 2\delta$ and the star topology graph. We note that the same scenario can be demonstrated in every graph that includes a node with the degree δ. Thus, we can establish a general proof for $\tau < \max\{2\delta, \chi(G^2)\}$ using the arguments in the proof of Lemma 2, where χ_2 is the chromatic number when considering distance-2 colouring.

Lemma 2. *Let $\xi \in \mathbb{R}, \tau \in \mathbb{N}$ and $S := \{[a\xi, (a+1)\xi) : a \in \{0, \ldots, \tau - 1\}\}$ be a partition of $[0, \xi\tau)$. The intervals $C := \{[b_i, b_i + \xi) : b_i \in \mathbb{R}\}_i$ intersects maximum $2|C|$ elements of S.*

Proof. Suppose that $[b, b + \xi) \in C$ intersects $I := [a\xi, (a + 1)\xi) \in S$ for some a. Either

$I = [b, b + \xi)$, $b \in I$ or $b + \xi \in I$. Therefore, any element $[b_i, b_i + \xi)$ of C intersects maximum 2 elements of S, one that contains b_i and one that contains $b_i + \xi$. □

2.4 Probabilistic stabilising TDMA Allocation and Alignment Algorithm

We propose Algorithm 1 as a self-stabilising algorithm for the $\mathcal{T}_{\text{TDMA}}$ task. The nodes transmit data packets, as well as control packets. Data packets are sent by active nodes during their data packet timeslots. The passive nodes listen to the active ones and do not send data packets. Both active and passive nodes use control packets, which include the reception time and the sender of recently received packets from direct neighbours. Each node aggregates the frame information it receives. It uses this information for avoiding collisions, acknowledging packet transmission and resolving hidden node problems. A passive node, p_i, can become active by selecting random timeslots, s_i, that are not used by active nodes. Then p_i sends a control packet in s_i and waiting for confirmation. Once p_i succeeds, it becomes an active node that uses timeslot s_i for transmitting data packets. Node p_i becomes passive whenever it learns about conflicts with nearby nodes, e.g., due to a transmission failure.

The hidden node problem refers to cases in which node p_i has two neighbours, $p_j, p_k \in \delta_i$, that use intersecting timeslots. The algorithm uses random back off techniques for resolving this problem. The passive nodes count a random number of unused timeslots before transmitting a control packet. The active nodes use their clocks for defining frame numbers. They count down only during TDMA frames whose numbers are equal to s_i, where $s_i \in [0, \tau - 1]$ is p_i's data packet timeslot. These back off processes connect all direct neighbours and facilitate clock synchronization, timeslot alignment and timeslot assignment. During legal executions, in which all nodes are active, there are no collisions and each node transmits one control packet once every τ frames.

Algorithm details The node status, $status_i$, is either active or passive. When it is active, variable s_i contains p_i timeslot number.

The frame information is the set $FI_i := \{id_k, type_k, occurrence_k, rxTime_k\}_k \subset \mathcal{FI} = \mathsf{ID} \times \{\mathsf{message}, \mathsf{welcome}\} \times \{\mathsf{remote}, \mathsf{local}\} \times \mathbb{N}$ that contains information about recently received packets, where $\mathsf{ID} := \{\perp\} \cup \mathbb{N}$ is the set of possible ids and the null value denoted by $\perp$. An element of the frame information contains the id of the sender id_k. The type $type_k = \mathsf{message}$ indicates that the sender was active. For a passive sender $type_k = \mathsf{welcome}$ indicates that there was no known conflict when this element was added to the local frame information. If $occurrence_k = \mathsf{local}$, the corresponding packet was received by p_i, otherwise it was copied from a neighbour. The reception time $rxTime_k$ is the time when this packet was received, regarding the local clock C_i, i.e., it is updated whenever the local clock is updated. The algorithm considers the frame information to select an unused timeslot. An entry in the frame information with timestamp t covers the time interval $[t, t + \xi)$.

Nodes transmit control packets according to a random back off strategy for collision avoidance. The passive node, p_i, chooses a random back off value, stores it in the variable

Algorithm 1: Probibilistic stabilising TDMA Allocation, code for node p_i

$status_i \in \{\mathsf{active}, \mathsf{passive}\}$; /* current node status */
$s_i \in \{0, \ldots, \tau - 1\}$; /* current data packet timeslot */
$wait_i, waitAdd_i \in \{0, \ldots, maxWait\}$; /* current back off countdown */
$FI_i := \{id_k, type_k, occurrence_k, rxTime_k\}_k \subset \mathcal{FI}$; /* frame information */
$BackOff() :=$ **let** $(tmp, r) \leftarrow (waitAdd_i,\ rnd(\{1, \ldots, 3\Delta\}))$; **return** $(\tau + r + tmp,\ 3\Delta - r)$;
/* reset backoff counter */
$\text{frame}() := (\text{GetClock}() \div \xi\tau) \bmod \tau$,; /* the current frame number */
$\text{Slot}(t) := (t \div \xi \bmod \tau), s() := \text{Slot}(\text{GetClock}())$; /* slot number of time t */
$\text{Local}(set) := \{\langle \bullet, \mathsf{local}, \bullet \rangle \in set\}$; /* dist-1 neighbours in set */
$\text{Used}(set) := \bigcup_{\langle \bullet, t_k \rangle \in set} \{\text{Slot}(t_k), \ldots, \text{Slot}(t_k + \xi - 1)\}$;
$\text{Unused}(set) := \{0, \ldots, \tau - 1\} \setminus \text{Used}(set)$; /* set of (un)used slots */
$\text{ConflictWithNeighbors}(set) := (\nexists_{\langle id_i \bullet \rangle \in set} \vee\ s_i \in \{\text{Slot}(t_i), \ldots, \text{Slot}(t_i + \xi)\} \vee$
$\exists_{\langle k, \bullet, rxTime \rangle \in set, k \neq id_i} : s_i \in \{\text{Slot}(rxTime - t_j + t_i), \ldots, \text{Slot}(rxTime - t_j + t_i + \xi)\})$; /* check for conflicts */
$\text{AddToFI}(set, o) := FI_i \leftarrow FI_i \cup \{\langle x, y, \mathsf{remote}, z' \rangle : \langle x, y, \bullet, z \rangle \in set, z' :=$
$(z + \max\{0, o\}) \bmod \mathcal{C}, z' \leq_{(\tau+1)\xi} C_i\}$; /* $set + FI_i$ */
$\text{IsUnused}(s) := s \in \text{Unused}(FI_i) \vee (\text{Unused}(FI_i) = \emptyset \wedge s \in \text{Unused}(\text{Local}(FI_i)))$; /* is s an unused slot? */

1.1 **upon** *timeslot*() **do**
1.2 **if** $s() = s_i \wedge status_i = \mathsf{active}$ **then** /* send data packet */
transmit$(\langle status_i, \text{Local}(FI_i), \text{MACfetch}() \rangle)$
1.3 **else if** $\neg(status_i = \mathsf{active} \wedge frame() \neq s_i)$ **then** /* check if our frame */
1.4 **if** $\text{IsUnused}(s()) \wedge wait_i \leq 0$ **then** /* send control packet */
1.5 **transmit**$(\langle status_i, \text{Local}(FI_i), \perp \rangle)$;
$\langle wait_i, waitAdd_i \rangle \leftarrow BackOff()$; /* next control packet countdown */
1.6 **if** $status_i \neq \mathsf{active}$ **then** $\langle s_i, status_i \rangle \leftarrow \langle s(), \mathsf{active} \rangle$;
1.7 **else if** $wait_i > 0 \wedge \text{IsUnused}((s() - 1) \bmod \tau)$ **then** /* count down */
$wait_i \leftarrow \max\{0, wait_i - 1\}$
1.8 $FI_i \leftarrow \{\langle \bullet, rxTime \rangle \in FI_i : rxTime \leq_{(\tau+1)\xi} \text{GetClock}()\}$; /* remove old entries from FI_i */

1.9 **upon** $\langle j, t_j, t_i, \langle status_j, FI_j, m' \rangle \rangle \leftarrow$ **receive**() **do**
1.10 **if** $\text{ConflictWithNeighbors}(FI_j) \wedge status_i = \mathsf{active}$ **then** /* conflicts? */
$\langle \langle wait_i, waitAdd_i \rangle, status \rangle \leftarrow \langle BackOff(), \mathsf{passive} \rangle$; /* get passive */
1.11 **if** $status_j = \mathsf{active}$ **then** /* active node acknowledge */
1.12 **if** $m' \neq \perp$ **then** $FI_i \leftarrow \{\langle id_i, \bullet \rangle \in FI_i : id_i \neq j\} \cup \{\langle j, \mathsf{message}, \mathsf{local}, t_i \rangle\}$;
1.13 **else if** $t_j = t_i \wedge \text{Slot}(t_j) \notin \text{Used}(FI_i)$ **then** /* passive node acknowledge */
1.14 $FI_i \leftarrow \{\langle id_k, \bullet \rangle \in FI_i : id_k \neq j\} \cup \{\langle j, \mathsf{welcome}, \mathsf{local}, t_i \rangle\}$;
1.15 **if** $t_i < t_j$ **then** /* converge-to-the-max */
1.16 $\text{AdvanceClock}(t_j - t_i)$; /* adjust clock */
1.17 $FI_i \leftarrow \{\langle \bullet, (rxTime + t_j - t_i) \bmod c \rangle : \langle \bullet, rxTime \rangle \in FI_i\}$; /* shift timestamps in FI_i */
1.18 $\langle \langle wait_i, waitAdd_i \rangle, status_i \rangle \leftarrow \langle BackOff(), \mathsf{passive} \rangle$; /* get passive */
1.19 $\text{AddToFI}(FI_j, t_i - t_j)$; /* Aggregate information on used timeslots */
1.20 **if** $m' \neq \perp$ **then** $\text{MACdeliver}(m')$;

$wait_i$, and uses $wait_i$ for counting down the number of timeslots that are available for transmissions. When $wait_i = 0$, node p_i uses the next unused timeslot according

to its frame information. During back off periods, the algorithm uses the variables $wait_i$ and $waitAdd_i$ for counting down to zero. The process starts when node p_i assigns $wait_i \leftarrow waitAdd_i + r$, where r is chosen uniformly by random from the set $\{1, \ldots, 3\Delta\}$, and updates $waitAdd_i \leftarrow 3\Delta - r$, cf. *BackOff*().

The node clock is the basis for the frame and timeslot starting times, cf. frame(), and respectively, $s()$, and also for a given timeslot number, cf. Slot(t). When working with the frame information, *set*, it is useful to have restriction by local occupancies, cf. Local(*set*) and to list the sets of used and unused timeslots, cf. Used(*set*), and respectively, Unused(*set*). We check whether an arriving frame information, *set*, conflicts with the local frame information that is stored in FI_i, cf. ConflictWithNeighbors(*set*), before merging them together, cf. AddToFI(*set*, *offset*), after updating the timestamps in *set*, which follow the sender's clock.

Node p_i can test whether the timeslot number s is available according to the frame information in FI_i and p_i's clock. Since Algorithm 1 complements the studied lower bound (Section 2.3), the test in IsUnused(s) checks whether FI_i encodes a situation in which there are no unused timeslots. In that case, IsUnused(s) tests whether we can say that s is unused when considering only transmissions of direct neighbours. The algorithm relies on the case in which $\tau > \max\{4\delta, \Delta + 1\}$. We use Lemma 2 here to argue that there are always a free time slot to communicate with neighbours.

The code of Algorithm 1 considers two events: (1) periodic timeslots (line 1.1) and (2) reception of a packet (line 1.9).

(1) *timeslot*(), *line 1.1:* Actives nodes transmit their data packets upon their timeslot (line 1.2). Passive nodes transmit control packets when the back off counter, $wait_i$, reaches zero (line 1.5). Note that passive nodes count only when the local frame information says that the previous timeslot was unused (line 1.7). Active nodes also send control packets, but rather than counting all unused timeslots, they count only the unused timeslots that belong to frames with a number that matches the timeslot number, i.e., frame() $= s_i$ (line 1.3).

(2) ***receive***(), *line 1.9:* Active nodes, p_i, become passive when they identify conflicts in FI_j between their data packet timeslots, s_i, and data packet timeslots, s_j of other nodes $p_j \in \Delta_i$ (line 1.10). When the sender is active, the receiver records the related frame information. Note that the payload of data packets is not empty in line 1.12, c.f., $m' \neq \perp$. Passive nodes, p_j, aim to become active. In order to do that, they need to send a control packet during a timeslot that all nearby nodes, p_i, view as unused, i.e., Slot(t) $\notin$ Used(FI_i), where t is the packet sending time. Therefore, when the sender is passive, and its data packet timeslots are aligned, i.e., $t_i = t_j$, node p_i welcomes p_i's control packet whenever Slot(t_j) $\notin$ Used(FI_i). Algorithm 1 uses a self-stabilising clock synchronization algorithm that is based on the converge-to-the-max principle. When the sender clock value is higher (line 1.15), the receiver adjusts its clock value and the timestamps in the frame information set, before validating its timeslot, s_i, (lines 1.16 to 1.18). The receiver can now use the sender's frame information and payload (lines 1.19 to 1.20).

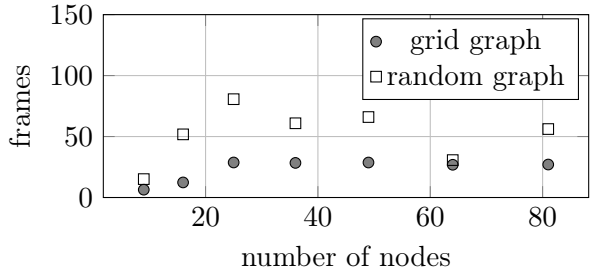


Figure 2.2: The convergence time in frames for different graphs. In the grid graph, nodes are placed on a lattice and connected to their four neighbours. The convergence times are the average over 16 runs that start each with random clock offsets. The random node graph is a unified disk graph with random node placement with maximal 16 neighbours pair node.

2.5 Experimental results

We demonstrate the implementation feasibility. We study the behaviour of the proposed algorithm in a simulation model that takes into account timing uncertainties. Thus, we demonstrate feasibility in a way that is close to the practical realm.

The system settings (Section 2.2) assumes that any (local) computation can be done in zero time. In contrast to this, the simulations use the TinyOS embedded operating systems [19] and the Cooja simulation framework [24] for emulating wireless sensor nodes together with their processors. This way Cooja simulates the code execution on the nodes, by taking into account the computation time of each step. We implemented the proposed algorithm for sensor nodes that use IEEE 802.15.4 compatible radio transceivers. The wireless network simulation is according to the system settings (Section 2.2) is based on a grid graph with $4 \geq \delta$ as an upper bound on the node degree and a random graph with $16 \geq \delta$ as an upper bound on the node degree. The implementation uses clock ticks of 1 millisecond. We use a time slot size of $\xi = 20$ clock ticks, where almost all of this period is spent on transmission, packet loading and offloading. The frame size is $\tau = 16 \geq 4\delta$ time slots for the grid graph and $\tau = 64 \geq 4\delta$ for the random graphs. For these settings, all experiments showed convergence, see Figure 2.2.

2.6 Conclusions

In Section 2.5, we presented experimental results that are based on the concept of Algorithm 1. Motivated by these results, we claim the following theorem.

Theorem 1. *Algorithm 1 is a probabilistic self-stabilising implementation of task* $\mathcal{T}_{\text{TDMA}}$.

The proof of Theorem 1 starts by showing the existence of unused timeslots by considering the case in which $\tau > \max\{4\delta, \Delta + 1\}$. For this, we use Lemma 2. This facilitates

the proof of network connectivity, as a success probability for control packets on each edge, clock synchronization (similar to [12]) and bandwidth allocation (Based on the fact that there are at least twice as many timeslots as competing nodes).

2.7 Acknowledgements

This work would not have been possible without the contribution of Olaf Landsiedel and Mohamed H. Mustafa in many helpful discussions, ideas, problem definition and analysis.

Bibliography

[1] Norman Abramson. "Development of the ALOHANET". In: *Info. Theory, IEEE Trans. on* 31.2 (1985), pp. 119–123.

[2] M. Arumugam and S.S. Kulkarni. "Self-stabilizing deterministic time division multiple access for sensor networks". In: *AIAA Journal of Aerospace Computing, Info., and Comm. (JACIC)* 3 (2006), pp. 403–419.

[3] Jean R. S. Blair and Fredrik Manne. "An efficient self-stabilizing distance-2 coloring algorithm". In: *Theor. Comput. Sci.* 444 (2012), pp. 28–39.

[4] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. "Contention-free MAC protocols for asynchronous wireless sensor networks". In: *Distributed Computing* 21.1 (2008), pp. 23–42. DOI: 10.1007/s00446-007-0053-x. URL: http://dx.doi.org/10.1007/s00446-007-0053-x.

[5] Hector Agustin Cozzetti and Riccardo Scopigno. "RR-Aloha+: A slotted and distributed MAC protocol for vehicular communications". In: *Vehicular Networking Conference (VNC), 2009 IEEE*. 2009, pp. 1 –8. DOI: 10.1109/VNC.2009.5416375.

[6] Praveen Danturi, Mikhail Nesterenko, and Sébastien Tixeuil. "Self-stabilizing philosophers with generic conflicts". In: *ACM Tran. Autonomous & Adaptive Systems (TAAS)* 4.1 (2009).

[7] Murat Demirbas and Muzammil Hussain. "A MAC Layer Protocol for Priority-based Reliable Multicast in Wireless Ad Hoc Networks". In: *BROADNETS*. IEEE, 2006.

[8] S. Devismes, S. Tixeuil, and M. Yamashita. "Weak vs. Self vs. Probabilistic Stabilization". In: *2008 The 28th International Conference on Distributed Computing Systems*. 2008, pp. 681–688. DOI: 10.1109/ICDCS.2008.12.

[9] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000. ISBN: 0-262-04178-2.

[10] Ted Herman. "Models of Self-Stabilization and Sensor Networks". In: *IWDC*. Ed. by Samir R. Das and Sajal K. Das. Vol. 2918. LNCS. Springer, 2003, pp. 205–214. ISBN: 3-540-20745-7.

[11] Ted Herman and Sébastien Tixeuil. "A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks". In: *Algorithmic Aspects of Wireless Sensor Networks: First International Workshop, ALGOSENSORS 2004, Turku, Finland, July 16, 2004. Proceedings*. Vol. 3121. LNCS. Springer, 2004, pp. 45–58. ISBN: 3-540-22476-9. DOI: 10.1007/978-3-540-27820-7_6. URL: http://dx.doi.org/10.1007/978-3-540-27820-7_6.

[12] Ted Herman and Chen Zhang. "Best Paper: Stabilizing Clock Synchronization for Wireless Sensor Networks". In: *SSS*. Ed. by Ajoy Kumar Datta and Maria Gradinariu. Vol. 4280. LNCS. Springer, 2006, pp. 335–349. ISBN: 978-3-540-49018-0.

[13] Amos Israeli and Marc Jalfon. "Token Management Schemes and Random Walks Yield Self-stabilizing Mutual Exclusion". In: *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*. PODC '90. Quebec City, Quebec, Canada: ACM, 1990, pp. 119–131. ISBN: 0-89791-404-X. DOI: 10.1145/93385.93409. URL: http://doi.acm.org/10.1145/93385.93409.

[14] Arshad Jhumka and Sandeep S. Kulkarni. "On the Design of Mobility-Tolerant TDMA-Based Media Access Control (MAC) Protocol for Mobile Sensor Networks". In: *Distributed Computing and Internet Technology, 4th International Conference, ICDCIT 2007, Bangalore, India, December 17-20, Proceedings*. Ed. by Tomasz Janowski and Hrushikesha Mohanty. Vol. 4882. LNCS. Springer, 2007, pp. 42–53. ISBN: 978-3-540-77112-8. DOI: 10.1007/978-3-540-77115-9_4. URL: http://dx.doi.org/10.1007/978-3-540-77115-9_4.

[15] Sandeep S. Kulkarni and Mahesh Arumugam. "Transformations for write-all-with-collision model› ". In: *Computer Communications* 29.2 (2006), pp. 183–199.

[16] Pierre Leone, Marina Papatriantafilou, and Elad Michael Schiller. "Relocation Analysis of Stabilizing MAC Algorithms for Large-Scale Mobile Ad Hoc Networks". In: *5th Inter. Workshop Algo. Wireless Sensor Net. (ALGOSENSORS)*. 2009, pp. 203–217.

[17] Pierre Leone, Marina Papatriantafilou, Elad Michael Schiller, and Gongxi Zhu. "Chameleon-MAC: Adaptive and Self-⋆ Algorithms for Media Access Control in Mobile Ad Hoc Networks". In: *12th Inter. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*. 2010, pp. 468–488.

[18] Pierre Leone and Elad Michael Schiller. "Self-Stabilizing TDMA Algorithms for Dynamic Wireless Ad-hoc Networks". In: *Int. J. Distributed Sensor Networks* 639761 (2013).

[19] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, et al. "TinyOS: An operating system for sensor networks". In: *Ambient intelligence*. Springer, 2005, pp. 115–148.

[20] Toshimitsu Masuzawa and Sébastien Tixeuil. "On bootstrapping topology knowledge in anonymous networks". In: *ACM Trans. Auton. Adapt. Syst.* 4.1 (2009), 8:1–8:27. ISSN: 1556-4665. DOI: 10.1145/1462187.1462195. URL: http://doi.acm.org/10.1145/1462187.1462195.

[21] Nathalie Mitton, Eric Fleury, Isabelle Guérin Lassous, Bruno Sericola, and Sébastien Tixeuil. "Fast Convergence in Self-Stabilizing Wireless Networks". In: *12th Int. Conf. Parallel and Distributed Systems (ICPADS'06)*. 2006, pp. 31–38.

[22] Michael Molloy and Mohammad R. Salavatipour. "A bound on the chromatic number of the square of a planar graph". In: *J. Comb. Theory, Ser. B* 94.2 (2005), pp. 189–213.

[23] Mohamed Mustafa, Marina Papatriantafilou, Elad Michael Schiller, Amir Tohidi, and Philippas Tsigas. "Autonomous TDMA Alignment for VANETs". In: *76th IEEE Vehicular Technology Conf. (VTC-Fall'12)*. IEEE, 2012, pp. 1–5. ISBN: 978-1-4673-1880-8.

[24] Fredrik Osterlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt. "Cross-level sensor network simulation with cooja". In: *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*. IEEE. 2006, pp. 641–648.

[25] Stéphane Pomportes, Joanna Tomasik, Anthony Busson, and Véronique Vèque. "Self-stabilizing Algorithm of Two-Hop Conflict Resolution". In: *12th Inter. Symp. on Stabilization, Safety, and Security of Distributed Systems (SSS'10)*. 2010, pp. 288–302.

[26] Riccardo Scopigno and Hector Agustin Cozzetti. "Mobile Slotted Aloha for Vanets". In: *Proceedings of the 70th IEEE Vehicular Technology Conference, VTC Fall 2009, 20-23 September 2009, Anchorage, Alaska, USA*. IEEE, 2009, pp. 1–5. DOI: 10.1109/VETECF.2009.5378792. URL: http://dx.doi.org/10.1109/VETECF.2009.5378792.

[27] Volker Turau and Christoph Weyer. "Randomized Self-stabilizing Algorithms for Wireless Sensor Networks". In: *IWSOS/EuroNGI*. Ed. by Hermann de Meer and James P. G. Sterbenz. Vol. 4124. LNCS. Springer, 2006, pp. 74–89. ISBN: 3-540-37658-5.

[28] Saira Viqar and Jennifer L. Welch. "Deterministic Collision Free Communication Despite Continuous Motion". In: *5th Inter. Workshop Algo. Wireless Sensor Net. (ALGOSENSORS)*. 2009, pp. 218–229.

[29] Fan Yu and Subir Biswas. "Self-Configuring TDMA Protocols for Enhancing Vehicle Safety With DSRC Based Vehicle-to-Vehicle Communications". In: *Selected Areas in Communications, IEEE Journal on* 25.8 (2007), pp. 1526 –1537. ISSN: 0733-8716. DOI: 10.1109/JSAC.2007.071004.

PAPER II

Olaf Landsiedel, Thomas Petig and Elad M. Schiller

DecTDMA: A Decentralized-TDMA with Link Quality Estimation for WSNs

Appeard as extended abstract in the proceedings of
Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, SSS 2016
Lyon, France
November 7-10, 2016, pp. 231–247

3 DecTDMA: A Decentralized-TDMA with Link Quality Estimation for WSNs

In wireless sensor networks (WSNs), different motes may transmit packets concurrently, i.e., having overlapping transmission periods. As a result of this contention, there are no packet reception guarantees and significant bandwidth can be lost. This contention can have a strong impact on the performance together with other kinds of interference sources, e.g., ambient noise. As a result, WSN's connectivity tends to have a very dynamic nature.

In this paper, we devise DecTDMA (Decentralized-TDMA), a fully decentralized medium access controller (MAC) that significantly reduces contention. It is based on a self-stabilizing algorithm for time division multiple access (TDMA). This self-stabilizing TDMA algorithm uses no external assistance or external references, such as wireless access points (WAPs) and globally-synchronized clocks. We present the design and implementation of DecTDMA and report encouraging results: our Cooja simulations and Indriya testbed experiments show stable connectivity and high medium utilization in both single and multi-hop networks. Since DecTDMA has favorable characteristics with respect to connection stability, we show that common link quality estimation (LQE) techniques further improve the operation of DecTDMA in the dynamic environment of low-power wireless networks.

3.1 Introduction

Wireless sensor networks (WSNs) are self-organizing systems where computing devices, so called motes (or nodes), set up – by themselves – a networking infrastructure without relying on external assistance. In this paper, we focus on medium access control (MAC) in WSNs. We present DecTDMA (Decentralized-TDMA) — a fully-decentralized MAC protocol for WSNs. Our decentralized solution does not assume access to external references, such as wireless access points (WAPs), and individual nodes in DecTDMA do not have special tasks, such as acting as elected coordinators. In this work, we aim to mitigate one of the key sources of internal interference: concurrent transmissions by neighboring motes, which cause radio interferences. The event of concurrent transmissions refers to the occurrence of multiple transmissions, such that the periods of these transmissions overlap. Concurrent transmissions have a great impact on the throughput in WSNs. For example, they can reduce the packet reception ratio (PRR) [26]. DecTDMA uses a self-stabilizing algorithm [19] for time division multiple access (TDMA) that significantly reduces the occurrence of concurrent transmissions. In addition, we show that DecTDMA deals well with other causes of WSN dynamics, such as mote or link failure and wireless links of intermediate quality, i.e., links with a PRR between 10% and 90%. DecTDMA uses a link quality estimation (LQE) technique for estimating the PRR of both broadcasts and their respective acknowledgements. We use this elegant (lightweight) and software-based technique for masking short term link failures, i.e., sporadic (receiver-side) packet omissions. It allows DecTDMA to sift out both disconnected links and (many) intermediate quality links. We present a TinyOS implemention of DecTDMA and evaluate it via Cooja simulations and experiments in the Indriya testbed [6]. During these experiments, we observe rather stable PRR values. Moreover, during our experiments, DecTDMA achieves PRR values that approach the analytical bounds.

Challenges In wireless communications, a single message may reach many receivers (due to the broadcast nature of radio transmissions). The success of message arrival depends on the distance between the transmitter and potential receivers as well as complex signal propagation patterns: wireless signals propagate unequally in different directions due to antenna characteristics, over many paths, and are subjected to interference. In WSNs, different (possibly neighboring) transmitters may send concurrently. In such cases, there are no guarantees for any receiver to get the packet and significant bandwidth can be lost. Thus, one of the key challenges in simplifying the use of WSNs is to limit the occurrence of such local contention factors. The studied question is whether one can device a MAC protocol that avoids contention, i.e., significantly reduces the occurrence of concurrent transmissions. We present DecTDMA and report encouraging results about the feasibility of TDMA protocols that require no external references, such as WAPs and global clocks.

Evaluation criteria Medium access control with high throughput is essential for many WSN protocols, especially for routing protocols [2]. Using statistical characterization, methods for link quality estimation (LQE) provide insights for routing protocols on which links they should forward packets. By avoiding low and intermediate quality links, the MAC layer can limit its packet loss, reduce the number of re-transmissions, and

provide better connectivity. This also impacts the higher layers, e.g., it reduces the need for selecting new routes at the network layer and reduce the end-to-end latency of the transport layer.

PRR is a common evaluation criterion in (wireless) communication networks [2]. For WSNs, PRR is often an elegant criterion that is easy to implement and according to which routing protocols can estimate link quality. In this work, we are interested both in the PRR values themselves and their stability over time. The motivation for the latter criterion is by the fact that stable PRR values are easier to work with (from the point of view of the higher layers).

The literature refers to PRR both as an evaluation criterion and as a basic mechanism for evaluating link quality. It is well-known that there are more advanced LQE mechanisms that provide more stable estimation than the average PRR mechanism [2]. We follow the common practice that often use average PRR mechanisms for simplicity and choose an LQE mechanism that considers the PRR values of both messages and their acknowledgements.

Design criteria In this paper, we focus on MAC protocols for low-power wireless networks that autonomously set up their networking infrastructure. WSNs are subject to faults that occur due to temporal hardware or software malfunctions or the dynamic nature of low-power wireless communications.

Fault-tolerant systems that are self-stabilizing [7] can recover after the occurrence of transient faults. These faults can cause an arbitrary corruption of the system state (as long as the code of the program is still intact). Transient faults can also represent temporary violations of the assumptions according to which the communication links and the entire dynamic networks behaves during normal operation. In order to provide DecTDMA with properties of self-organization and self-recovery, we base the implementation of DecTDMA on an existing self-stabilizing TDMA algorithm [19]. This algorithm helps DecTDMA to significantly reduce the occurrence of concurrent transmissions. We note that the design of the TDMA algorithm in [19] focuses on packet loss due to concurrent transmissions and models all other kinds of packet loss as transient faults (after which a brief recovery period is needed). DecTDMA extends this, by utilizing an LQE technique for masking sporadic packet loss. Thus, DecTDMA considers fewer occurrences of sporadic packet loss as transient faults than the TDMA algorithm by Petig et al [19]. As a result, DecTDMA avoids unnecessary recovery periods. This increases the performance and the stability of the packet reception ratio (PRR) values.

Our design criteria of self-organization and self-recovery simplify the use of WSNs. It reduces the effect of local and low-level complications, such as contention management, that many systems leave to be handled by the higher layers. Consequently, we provide an important level of abstraction that allows the higher layers to focus on their tasks, such as routing table construction, packet forwarding, and end-to-end communications.

We do not claim that the studied implementation is self-stabilizing. Note that the implementation of a self-stabilizing system requires every system element to be self-stabilizing [3], rather than just a subset of the needed algorithms. This includes the use of compliers that preserve any invariant that is related to the corretness proof [9], as well as the use of self-stabilazing CPUs [8], self-stabilazing operating systems [10] to name a

few.

Our Contribution In this paper, we present DecTDMA — a decentralized TDMA that does not assume access to external references, such as wireless access points (WAPs) or a global clock. For DecTDMA, we (a) extend and (b) implement an existing self-stabilizing algorithm [19] and (c) evaluate our implementation both via WSN simulations and a real-world testbed. By that, we provide stable connectivity with high values of packet reception ratio (PRR). We also offer a masking technique as a way to further improve the channel stability by considering sporadic packet losses as normal faults rather than transient ones. The studied technique estimates the stability of every TDMA time slot and lets DecTDMA to keep only time slots that are above a predefined threshold. We evaluate our TinyOS implementation via Cooja simulations on both single and multiple hop networks as well as on Indriya Testbed at NUS (with 97 motes). The results validate that DecTDMA (and its LQE technique for masking sporadic packet loss) provides stable connectivity with high PRR values. For the studied cases of network simulations, DecTDMA achieves PRR values that are rather stable and approach the analytical bounds.

We believe that these findings demonstrate the feasibility of decentralized reference-free TDMAs that provide stable communication among the WSN motes without the need for an external coordinator nor access to a global time reference. The design and implementation of DecTDMA exposes the advantage of following the self-stabilization design criteria. DecTDMA is a multifaceted TDMA protocol that can deal with a number of failures. This fault model includes concurrent transmissions and sporadic packet loss, as well as different violations of the algorithm assumptions, which we model as transient faults.

Document Structure As background knowledge (Section 3.2), we present the self-stabilazing TDMA algorithm by Petig et al. [19], which DecTDMA extends. We complete the description of DecTDMA by discussing the details of our LQE technique (Section 3.3). We present our evaluation of DecTDMA by studying the results of our experiments (Section 3.4). Finally, we discuss the related work (Section 3.5) and our concluding remarks (Section 3.6).

3.2 Background: Time Slot Alignment and Allocation

DecTDMA uses a TDMA algorithm by Petig et al. [19] for aligning the frame and letting each mote access a time slot that is unique within its 2-hop neighbourhood (Figure 3.1). For the sake of self-containment of this paper, we describe the algorithm and how it works under the assumptions of Petig et al. [19]. In real-world WSNs, there are different sources for interferences that cause packet loss. The TDMA algorithm of Petig et al. [19] focuses on packet losses that are due to concurrent transmissions and models all other kinds of packet losses as transient faults (after which brief recovery periods are needed). Therefore, DecTDMA extends it and uses an LQE technique for masking sporadic packet loss (Section 3.3). We show that DecTDMA can perform well in real-world WSNs, which do not need to follow the assumptions made by Petig et al. (Section 3.4).

In the TDMA algorithm by Petig et al. [19], motes send both data and control packets.

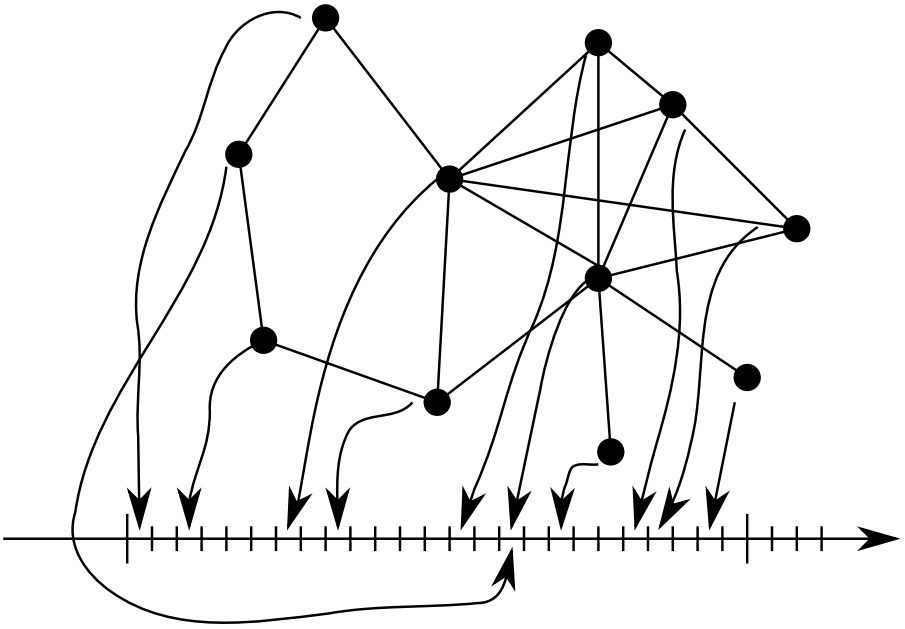


Figure 3.1: Time slot assignment in multi-hop network graphs. Neighbours with a distance of at least 3 can share the same time slot without collision.

Also, the motes can play an active or a passive role according to their (local) status. When the mote p_i's status is active, it sends data packets during a time slot that is designated for p_i's data packets, which we call p_i's transmission time slot, s_i. When p_i's status is passive, it listens to the active motes, and it does not send any data packets. The motes send, regardless of their status, control packets which include the frame information (FI). The field FI includes data about the recently received data packets from direct neighbours. That information refers both to the sender identity and the packet sending time. The TDMA algorithm by Petig et al. aggregates the frame information it receives during the past frame. The algorithm relies on FI to acknowledge transmissions, resolve hidden terminal phenomena and avoid concurrent transmissions. The mote active-passive status changes according to the filed FI.

(1) **The passive mote p_i takes the following steps in order to become active.** It selects a random time slot, s_i, that no active mote within two hops uses according to p_i's FI. Mote p_i tests the use of time slot s_i by sending a control packet and waiting for acknowledgement from neighbouring motes. These acknowledgements are included in their data and control packets. Whenever that test works, mote p_i changes its status to active and uses s_i as its transmission time slot for data packets.

(2) **The active mote p_i can become passive due to the following.** An active mote

p_i changes its status to passive after its FI field reports about another mote, p_j, that transmits during its time slot s_i, where p_j is at most two hops away from p_i. Recall that during the occurrence of hidden terminal phenomenon, mote p_i has a neighbour, p_j, and a distance two neighbour, p_ℓ, such that p_i and p_ℓ use time slots with overlapping periods. In this case, there is no grantee that p_i receives p_j's frame information (and acknowledgements) to p_i's packets. In order to deal with this issue, the TDMA algorithm by Petig et al. also considers the absence of p_j's acknowledgement as an implicit report on a possible occurrence of the hidden terminal phenomenon. Note that, unlike the TDMA algorithm by Petig et al., DecTDMA uses an LQE technique for mitigating the effect of sporadic packet loss, say, due to ambient noise. Namely, DecTDMA lets p_i change its status from active to passive only according to LQE indication (Section 3.3).

Petig et al. [19] uses a random back-off mechanism for dealing with contention scenarios in which "too many" passive motes are testing random time slots concurrently, see case (1) above. This mechanism counts down (from a randomly selected backoff value) every time the node observes a time slot that for which it receives no message. The TDMA algorithm by Petig et al. also adopts a technique for clock synchronization according to which it aligns the TDMA time slots. Petig et al. [19] show that their self-stabilizing TDMA algorithm can provide, after a convergence period, guarantees that each active mote can transmit successfully once in every TDMA frame. The proof of self-stabilization by Petig et al. assumes that packet loss occurs only due to concurrent transmissions. However, in real-world WSNs the above assumption does not hold. This work proposes DecTDMA, which does not follow this assumption. Instead, it uses an LQE technique for avoiding a change in p_i status from active to passive due to the occurrence of sporadic packet losses (and does allow this change whenever p_i's time slot, s_i, losses packets repeatedly, see Section 3.3).

3.3 TDMA Protocol with Link Quality Estimation

In real-world WSNs, packet losses occur due to many reasons, such as external interference or concurrent transmissions, i.e., when neighboring nodes transmit during overlapping periods. DecTDMA addresses the challenge of sporadic packet losses via a Link Quality Estimation (LQE) procedure. Here, mote p_i does not stop sending a data packet in its transmission time slot s_i due to a sporadic packet loss. We use a software-based LQE technique that estimates p_i's LQE by accumulating the acknowledgments that p_i receives over a time window and comparing them to the number of transmitted packets. We use this lightweight LQE technique for deciding whether p_i shall keep its transmission time slot, s_i, or try to randomly select a new one after a random back-off period (Section 3.2).

Our LQE technique considers a time window of w TDMA frames. We use arrays of integers, $ack_i[]$ and $rx_i[]$ (each of w entries), which in the beginning of every time window, we initialize each entry with the zero value. During each time window, when p_i receives a data packet during time slot s_j, it increments $rx_i[j]$. Moreover, if that packet includes

an acknowledgement for the packet p_i sent previously, it also increments $ack_i[j]$. At the end of each time window, p_i tests whether there is a time slot s_j for which $rx_i[j] \geq \mathcal{T}_{rx}$ (the reception threshold) and $ack_i[j] \leq \mathcal{T}_{ack}$ (the acknowledgement threshold). In case p_i finds such s_j, it stops using its transmission time slot, s_i. This process repeats in every time window during which p_i's status is active.

Note that we assume that the communication links have symmetrical packet loss behavior. We justify the packet reception and acknowledgement thresholds of $\mathcal{T}_{rx}$, and respectively, $\mathcal{T}_{ack}$ by considering a pair of neighbouring motes, p_i and p_j. Suppose that the successful transmission probability from p_i to p_j (and visa verse) is p. In a given time window of w frames, the expected number of p_i's packets that p_j receives is wp and the expected amount of acknowledgements is wp^2. During our experiments, we have selected a window of $w = 20$ TDMA frames, the reception threshold $\mathcal{T}_{rx} = 0.8w = 16$ by taking $p = 0.8$ and considering a value that includes all reliable links, which are defined as the ones that have 90% PRR [2]. We decided to consider $\mathcal{T}_{ack} = 0.4w = 8$ as the acknowledgement threshold since it presented a more stable behavior than $p^2 = 0.64$ during our experiments (Section 3.4).

3.4 Evaluation

We evaluate DecTDMA with respect to channel stability and throughput via the Cooja simulator and the Indriya Testbed at NUS with 97 motes. We implemented DecTDMA on top of TinyOS version 2.1.1. For the Cooja simulations, we select both single and multiple hop topologies whereas in the testbed experiments the focus is on the multiple hop case. Our results show a high throughput as well as acceptable channel stability performances of DecTDMA under real-world conditions. We also show that DecTDMA further improves the TDMA algorithm by Petig et al. [19] via the proposed mechanism for channel quality estimation.

Every WSN has a number of inherent uncertainties. In this dynamic environment, it is challenging for any node to maintain a stable rate of packet reception. This channel stability criterion is one of the important metrics, which we evaluate. Another important criterion is the throughput of DecTDMA, which considers the number of successful packet receptions. Note that for the simulation results, we normalize these numbers of successful packet receptions and compare them to an analytical maximum (which we tailor for each studied topology).

The TDMA frame setup We use the notation below when presenting our results. DecTDMA considers the case in which every node uses at most one time slot per TDMA frame for data packets. Despite this assumption, DecTDMA is obviously extendable by simply allowing each mote to use more than one time slot. We use τ to denote the number of time slots per TDMA frame and ξ to refer to the length of each time slot in seconds. Node that each mote can transmit at most $((1\ \mathtt{s}/\xi)/\tau)$ packets per second. In our frame setup, $\varphi = ((1\ \mathtt{s}/\xi)/\tau) = 2$ is an upper bound on the number of frame per seconds that each node uses (after convergence) for data packets in every second.

Single hop WSNs: simulation results Single hop WSNs represent the case in

which every mote can communicate directly with any other mote. The complete graph topology of these networks is absent from multi-hop dynamics that are due to, for example, fading signal strength. Moreover, this setup has a clear analytical upper bound of the throughput, i.e., in a network of n transmitters at most $n-1$ packets are received per transmission. In this basic setup, we are able to demonstrate that DecTDMA's throughput approaches the analytical upper bound. Note that even though this setup is simpler than all the others that we study, the presented performances are not straightforward, because our fully-decentralized implementation has no access to external assistance, such as access points, or external references, such as a global clock. Yet, we show that DecTDMA's performances are close to the analytical bounds.

We model a single-hop graph using the complete graph K_n. During the Cooja simulation, we use p as the transmission success probability when a packet is sent from a node to a neighboring one. We use these settings for evaluating how close can DecTDMA approach the analytical bounds, which depend on p. We normalize the number of received data packet per second using $\#pkts/T/(n-1)/(\varphi/n)$, where $\#pkts/T$ refers to the average number of received packets per second over a time period T and φ is the amount of data packets per node that we expect per second. Note that, $(n-1)/(\varphi/n)$ defines the expected number of received packets in K_n (if there is no packet loss). Therefore, $\#pkts/T/(n-1)/(\varphi/n)$ should approach p, when the packet reception probability is p.

The plot in Figure 3.2 presents the (normalized) number of received packets for different numbers of nodes when considering K_n, where $n \in \{5, 6, \ldots, 40\}$. The chart shows that DecTDMA behaves well when the transmission (and reception) success probability is $p = 1$. Note that this is the case in which we are running DecTDMA as an implementation of Petig et al., because the settings are similar to the ones that Petig et al. considered in [19]. Since Petig et al. do not consider the case of $p < 1$, DecTDMA version with LQE indeed out performs the one without. Thus, for the case of single hop networks, DecTDMA with LQE performs well for the case of $p < 1$ and for the case of $p = 1$, there is no need to use LQE as a masking technique (and the theoretical assumption that $p = 1$).

Multiple hop WSNs: simulation results The phenomenon of hidden terminal consider the case in which mote p_1 can communicate directly with both p_0 and p_2 but p_0 and p_2 cannot communicate directly (Figure 3.3). In this case, node p_0 is hidden from p_2 and thus the only way that it can identify that its transmissions occur concurrently with p_2 is via p_1 assistance. We consider a multiple hop setup, in which the hidden terminal phenomenon exists and yet we are able to compare between DecTDMA's throughput and an analytical upper bound that we tailor. Interestingly, these simulations show a behavior that is similar to the above single hop networks, which use the complete graph K_n.

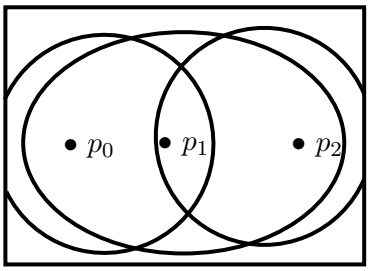


Figure 3.3: The hidden terminal.

We also consider settings for the a 2-hop graph $G_2(n) := (V, E)(n)$ with n vertices.

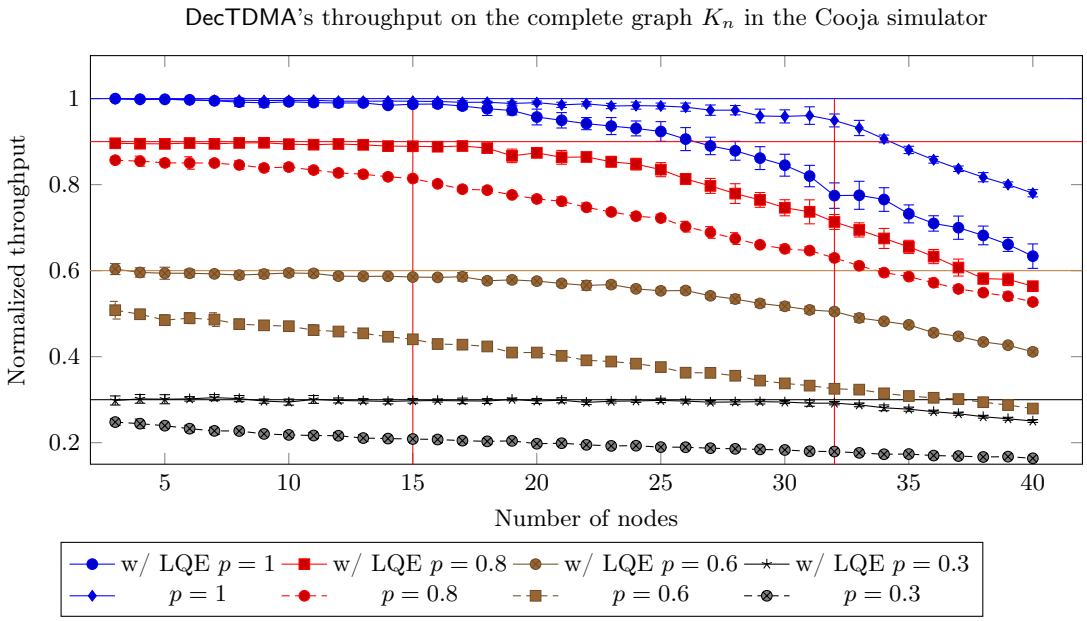


Figure 3.2: DecTDMA without and with LQE on the complete graph K_n. The probability of a successful transmission is p. The throughput is normalized by $\#pkts/T/(n-1)/2/n$. The (colourful) horizontal lines represent the analytical bounds on the throughput. The (red) vertical lines stand for the bounds for guaranteed convergence, which is 15, and the frame size, which is 32.

The set of vertices is partitioned in four sets S_0, S_1, S_2 and S_3, such that each set forms a clique in G_2 (Figure 3.4). We define a cardinally constraint that requires these cliques to be of similar size: $|S_{i+1}| + 1 \geq |S_i| \geq |S_{i+1}|$ for $i \in \{0, 1, 2\}$ and $|S_3| + 1 \geq |S_0|$. In addition to the edges within every vertex to any other vertex in its clique, we define an edge between every vertex and any other vertex that is in a neighboring clique. We say that clique S_i is neighboring to clique $S_{i+1 \bmod 4}$ and $S_{i-1 \bmod 4}$. Note that for the case of $n \bmod 4 = 0$, G_2 is regular, i.e., all vertexes have the same degree. During the Cooja simulation, we use p and $q = 0.4p$ as the transmission success probabilities when a packet is sent from a node to a neighboring one that shares, and respectively, does not share the same clique. In this rather simple settings for multiple hop networks, we can still evaluate how DecTDMA is close to analytical bounds that depend on p and $q = 0.4p$. Note that we study DecTDMA behavior on multiple hop graph using testbed experiments.

We normalize the throughput (Figure 3.5) by the expected throughput for the case there is no packet loss. The difference to the 1-hop case on the K_n is that from one clique, S_i, the opposite corner clique, $S_{i+2 \bmod 4}$, cannot be reached, thus the number of nodes we expect to received a packet is reduced by a quarter. This leads to $\#pkts/T/(\frac{3}{4}n - 1)/(\varphi/n)$. In Figure 3.6, we use a different probability for successful reception for the neighboring cliques. Since they contain half of the nodes and packets are received with probability $q = 0.4$ in case $p = 1$, we get the bound $(0.5nq + 0.25n - 1)/(\varphi/n)$. Note that

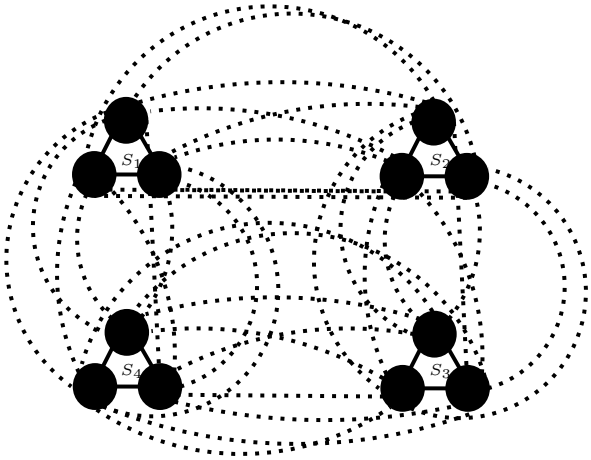


Figure 3.4: The $G_2(12)$ graph. We assume two probabilities for a sucessfull transmission, p and q. Nodes at the end of solid lines in this figure can successfully transmit packets with probability p and nodes at the end of dotted lines with probability q. The results are presented in figures 3.5 with $q = p$, and respectively, 3.6 with $q = 0.4p$.

the rate q is linear in p. Thus, q scales down for smaller values of p. This leads to the fact that for a given transmission success rate of p (within the clique), we also expect p to be the normalized throughput.

The plots in figures 3.5 and 3.6 present the simulation results for the $G_2(n)$ topology and $n \in \{5, 6, \ldots, 40\}$ when considering the cases in which the successful transmission probability of links that connects nodes that are at different cliques is $q = 0.4p$ and within a clique p (cf. Figure 3.4). We note the similarly of the results that appear in figures 3.2 and 3.5 even though the latter set of experiments refer to a two hop communication graph, rather than one hop, as in the former set. Moreover, when running DecTDMA with LQE, we observe an acceptable degree of stability in the number of packets received for every transmission. Of course, the values in Figure 3.6 are significantly less than the ones in Figure 3.5 (due to higher packet loss rate between neighboring cliques).

Multiple hop WSNs: testbed experiments We complete our evaluation of DecTDMA by running experiments in the Indriya Testbed at NUS [6]. This is a controllable environment and yet it is representative to real-world WSNs with respect to the actual interference that the deployed motes encounter, e.g., dynamic link behavior, say, with respect to PRR values [2]. Our experiments consider running DecTDMA over 97 motes that form a multipile hop network (Figure 3.8). Such real-world networks are known to have different and dynamic transmission success rates. We compare between the cases in which DecTDMA includes and does not include our LQE technique. The plot in Figure 3.8 shows the long term impact of ambient noise on DecTDMA with and without LQE. Whereas the former is able to improve over time by learning about the presence of links with low PRR values, the latter can spiral down due the fact the Petig et al. [19]

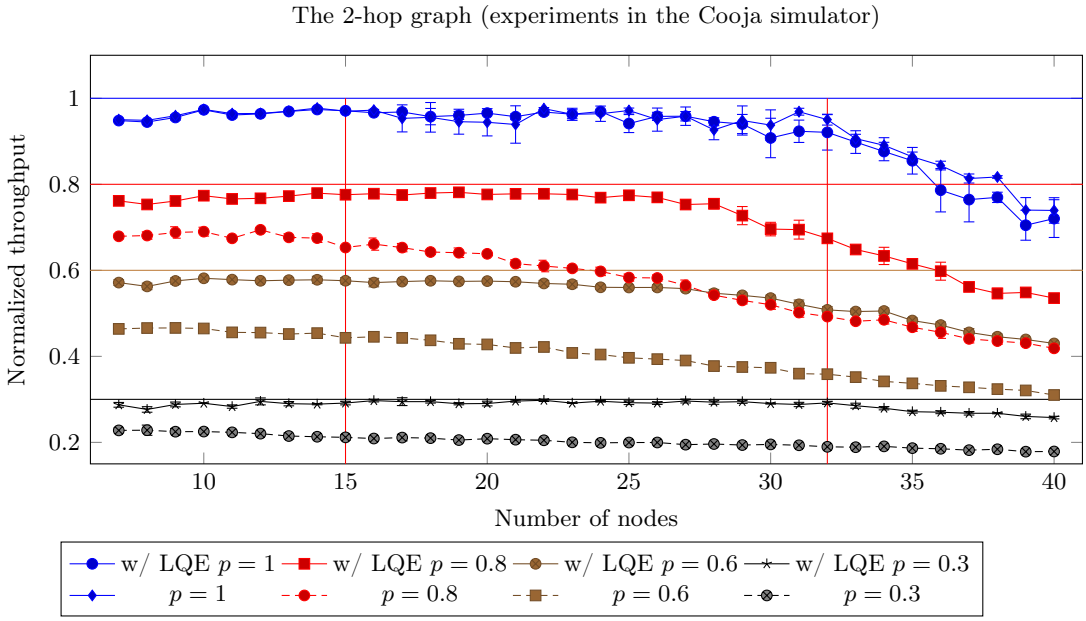


Figure 3.5: DecTDMA with and without LQE on the 2-hop graph $G_2(n)$ and uniform probability, p, of successful transmissions, where $n \in \{6, 7, \ldots, 40\}$ and $p \in \{0.3, 0.5, 0.8, 1.0\}$ is the probability for successful transmission between any pair of motes that can communicate directly. The horizontal (colourful) lines represent the analytical bounds and the (red) vertical lines the bounds for convergence (15) and the frame size (32). The throughput is normalized by $\#pkts/T/(0.75n - 1)/2/n$.

do not consider sporadic packet omission.

Since this work considers experiments that run both on the Cooja simulator and the Indriya testbed, we also wanted to run in Cooja experiments on a multiple hop graph that resembles the one that the Indriya testbed uses (Figure 3.7). Broadly speaking, the two plots in figures 3.7 and 3.8 resembles. We note that there is no clear recipe for Cooja to consider in detail the dynamics of real-world WSNs, such as the Indriya testbed. Hence, differences between these plots are inevitable. Also, there is no straightforward way to compare our results on the Indriya testbed to an analytical bound, as we did in figures 3.2, 3.5 and 3.6.

Evaluation summary DecTDMA with LQE presents high and stable throughput values in Cooja simulations (in single and multiple hop networks) that approach our analytical bounds. The Indriya testbed experiments show stability of the throughput values that resembles to the ones in the Cooja simulations.

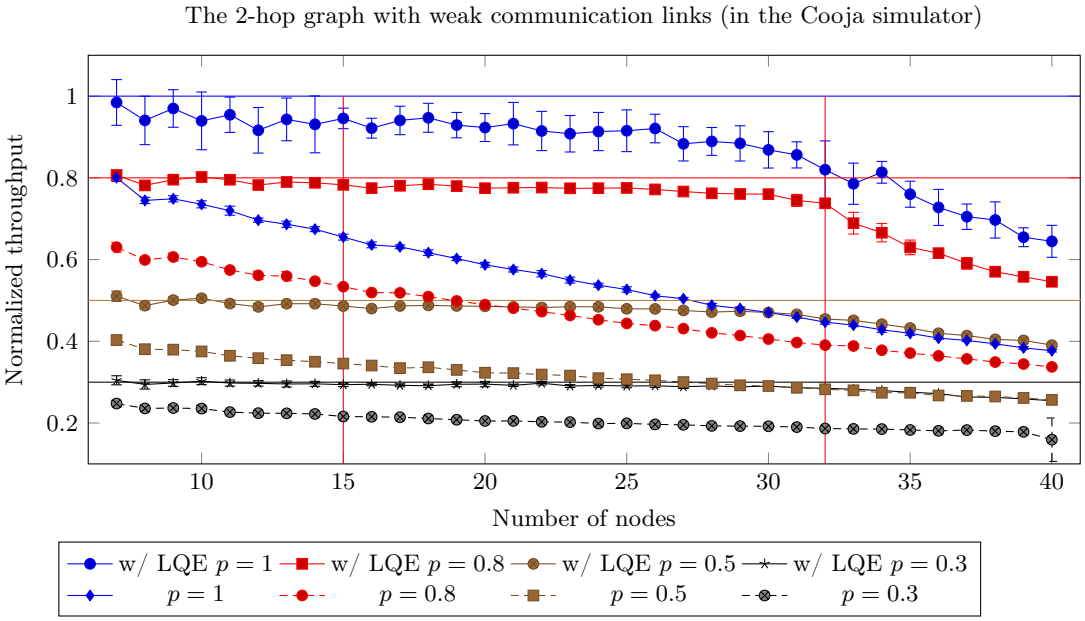


Figure 3.6: DecTDMA with and without LQE on the 2-hop graph $G_2(n)$, where $n \in \{6, 7, \ldots, 40\}$. The probability of successful transmissions between any two motes that belong to the same clique is $p \in \{0.3, 0.5, 0.8, 1.0\}$. The ones that belong to neighboring cliques have the probability of $q = 0.4p$. The horizontal (colourful) lines represent the analytical bounds, which depends on p, and the (red) vertical lines the bounds for convergence (15) and the frame size (32). The throughput is normalized by $\#pkts/T/(0.75n-1)/2/n$, as in Figure 3.5.

3.5 Related Work

ALOHAnet and its many variances [1] are MAC protocols that schedule the medium access randomly. Time division multiple access (TDMA) follows a scheduled approach that divides the radio time into TDMA frames and then further divides these frames into time slots. We note that at high and stable PRR values, TDMA protocols offer inherently a greater degree of predictability than the ones that access the medium randomly. The TDMA task that we consider in this work includes both the alignment of frames and time slots as well as the allocation of these time slots to the motes, rather than just the latter part of the task, as many existing TDMA protocols do.

Existing approaches for MAC-layer contention management consider priorities (for maintaining high bandwidth utilization while meeting the deadlines, such as [22]) or modifying the signal strength or carrier sense threshold [25]. We view both approaches as possible extensions to DecTDMA, which considers a single priority and does not adjust the radio settings dynamically. DecTDMA uses fixed size TDMA frames and it allocates TDMA time slots until saturation, i.e., no more time slots are left to allocate. Note that a number of techniques can prevent starvation in saturated situations, such as

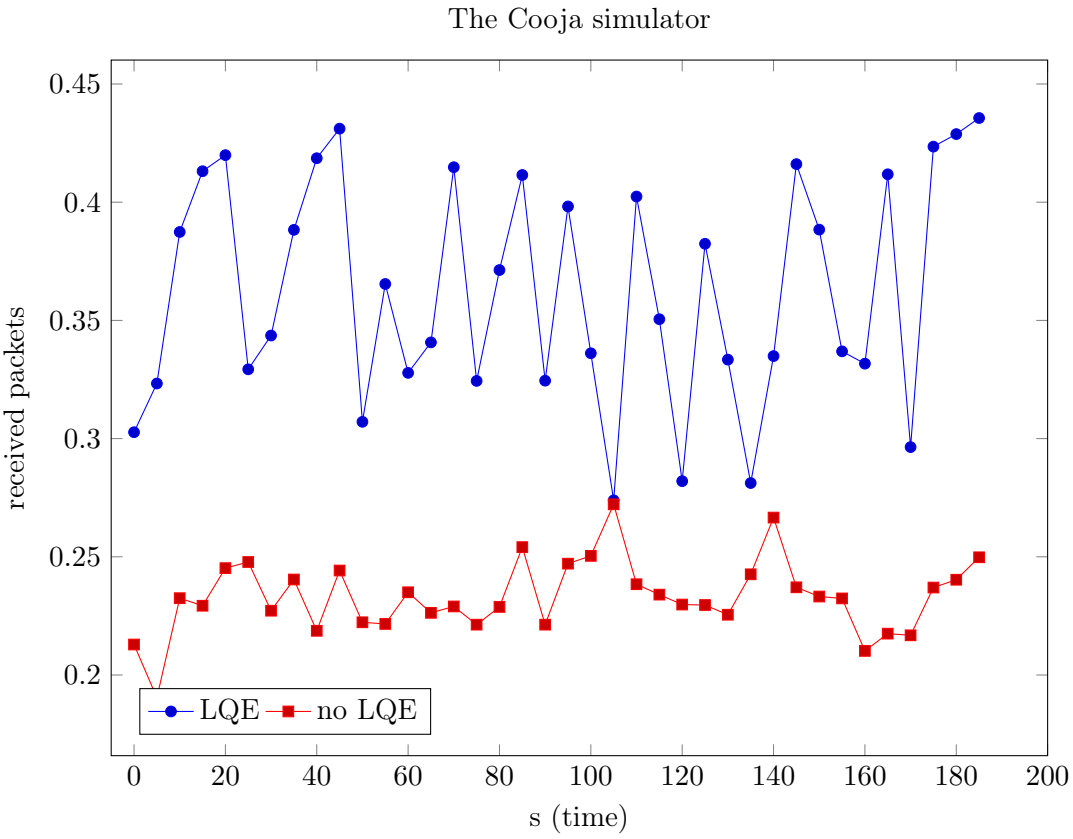


Figure 3.7: Comparison between TDMA with and without link quality estimation on the Cooja simulator. This plot shows the number of received packets accumulated in intervals of 100 s in Cooja. The overall time of the experiment is 200 s on the 2-hop graph $G_2(40)$.

limiting the number of TDMA frames that the application can use consecutively without deferring further communication. Once, we apply such techniques, the (common) back-off mechanism of DecTDMA will prevent starvation.

The literature considers receiver-side collision detection [25, 5], which requires hardware support for signal processing as well as receivers to notify senders about the success or failure of transmissions. In this paper, we assume hardware that does not support collision detection (on the sending side or on the receiving side). In DecTDMA, however, the payload does include a short summary of the frame information (FI) [19]. We prefer not to assume access to external references and provide a fully-decentralized implementation since unbounded signal failure can occur, e.g., in underground tunnels. STDMA [27] is an example of a protocol that assumes the availability of an external reference (GNSS [24]). It allocates bandwidth according to the position of motes.

The (self-stabilization) literature on TDMA algorithms often does not answer the causality dilemma of "which came first, synchronization or communication." On one

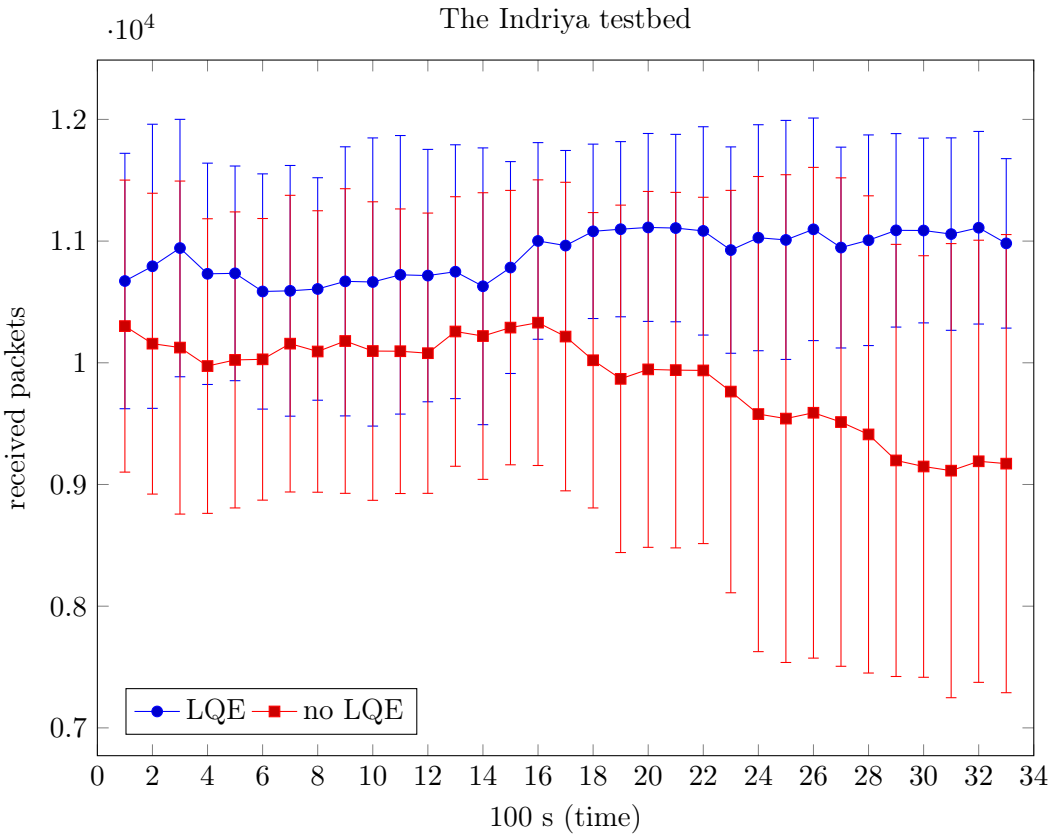


Figure 3.8: Comparison between TDMA with and without link quality estimation on the INDRIYA testbed [6]. This plot shows the number of received packets accumulated in intervals of 100 s as an average over 10 run each. The total run time for each run is 3600 s. The error bars represent in both graphs the standard deviation.

hand, existing clock synchronization algorithms often assume the existence of MAC algorithms that offer bounded communication delay. However, on the other hand, existing MAC algorithms that provide bounded communication delay, often assume access to synchronized clocks. For example, some TDMA protocols [21] assume the availability of a clock synchronization algorithm that can reach the needed clock synchrony bound before starting the allocation of time slots. Where there is no external reference or assistance, the implementation of TDMA protocol requires time slot alignment *during* the time slot allocation process. DecTDMA needs to address these tasks at the same time (without external reference). Busch et al. [4] and Petig et al. [19] propose TDMA algorithms that address the above challenge without assuming access to external references. Busch et al. [4] address this by assuming that the number of time slots in each frame is at least $2(\Delta + 1)$, where Δ is an upper bound on the number of nodes with whom any node can communicate with using at most one intermediate node for relaying packets. Petig et

al. [19] provides a solution that requires a frame size to could be $O(\sqrt{\Delta})$ times smaller. Moreover, they show that you cannot do much better than that. Schneider and Wattenhofer [23] present a local algorithm for vertex coloring that could be the basis a for self-organizing TDMA protocol. We chose to base DecTDMA on a TDMA algorithm [19] that considers self-stabilization explicitly. Other proposals for self-stabilizing MAC algorithms exist [11, 12, 16] as well as other algorithm cited by [19]. We choose the one that can deal with networks dynamics, assume no access to external reference, has a more attractive overhead (with respect to the frame information) and follows conventional TDMA practice, such as fix packet size.

The IEEE 802.15.4-2015 standard describes the DSME extension. This extension covers multi-hop TDMA and a slot allocation method. One possible implementation is OpenDSME [13] and [17]. The authors also provide a formal analysis of the time slot allocation [14] and an analytical model for large scale networks [18]. In contrast to our our work, requires DSME separate clock synchronisation, which is done by beacons that are transmitted by specific nodes. Another difference lays on the communication side, while we use a time slot to send a broadcast, is the time slot in DSME dedicated to pairs of nodes. This is motivated by the fact that IEEE 802.15.4 describes a full protocol stack for real-world applications which in cloud the routing of information between two nodes over multiple hops in the network. Our focus, in contrast, is on looking into bounds for embedding decentralised synchronisation and time slot assignment in one single radio channel.

We are not the first to consider the provision of improved link quality. One of the most notable examples is a line of research work that has started by Kuhn, Lynch and Newport [15], which presented the abstract MAC model. They propose a number of high-level communication primitives that use an (abstract) unreliable MAC layer, and yet provide guarantees with respect to the packet delivery to the application layer at the receiver-side, say, after a bounded number of retransmissions. We follow a complementary approach to the one of Kuhn, Lynch and Newport [15], because we are interested in possible guarantees with respect to the packet reception at the receiver-side without considering the possibility to retransmit a lost packet.

3.6 Discussion

Designing and implementing a MAC protocol for WSNs is a non-trivial task. The challenges include a various source of interferences as well as the need to recover rapidly from the occurrence of failures that these interferences cause. Using DecTDMA, we were able to exemplify how to have the self-stabilization design criteria in mind while designing a MAC protocol that works well in a real-world WSN testbed, such as Indriya [6].

This design process started with the self-stabilizing TDMA algorithm by Petig et al. [19]. That algorithm modelled, for example, the communication graph and the manner in which the motes exchange messages. The focus of Petig et al. is on dealing with one of the most destructive interferences in WSNs, which is packet loss due to concurrent transmissions. Petig et al. consider a fault model that includes concurrent transmissions

whereas sporadic packet losses, say, due to ambient noise, are considered as transient faults. This focus on concurrent transmissions allows, via a rigorous analysis, an exact design of their self-stabilizing TDMA algorithm. Our experiments validate that indeed, in the absence of transient faults, e.g., sporadic packet loss, the self-stabilizing TDMA algorithm Petig et al. addresses the challenge of avoiding concurrent transmissions (figures 3.2 and 3.5).

We present DecTDMA, which is a TDMA protocol that extends the fault model of Petig et al. and thus sporadic packet loss are no longer considered as transient faults. This paper shows that via an elegant LQE technique that masks the effect of sporadic packet loss, the PRR levels of DecTDMA are higher significantly than the ones of Petig el al. [19]. Moreover, we observe the stability of these PRR values also in a real-world testbeds, such as Indriya [6] (Figure 3.8).

This work shows how to deal with failures and interferences in non-trivial real-world challenges, such as the design of fully-decentralized reference-free TDMA protocol. Our design process enhanced iteratively the fault model during the design of Petig el al. [19] and then in this work, we used an elegant masking technique to further enhance the fault model. DecTDMA is a successful example of the above design and development process that have the self-stabilization design criteria in mind. As future work, we offer the reader to study real-world problems and use the presented design and development process.

Acknowledgments

We knowledge the participation of Henning Phan in this work by assisting the protocol implementation [20]. This work has been partially supported by the Swedish Energy Agency under the program Energy, IT and Design.

Bibliography

[1] Norman M. Abramson. "Development of the ALOHANET". In: *IEEE Trans. Information Theory* 31.2 (1985), pp. 119–123. DOI: 10.1109/TIT.1985.1057021. URL: http://dx.doi.org/10.1109/TIT.1985.1057021.

[2] Nouha Baccour et al. *Radio Link Quality Estimation in Low-Power Wireless Networks*. Springer Briefs in Electrical and Computer Engineering. Springer, 2013. ISBN: 978-3-319-00773-1. DOI: 10.1007/978-3-319-00774-8. URL: http://dx.doi.org/10.1007/978-3-319-00774-8.

[3] Olga Brukman, Shlomi Dolev, Yinnon A. Haviv, Limor Lahiani, Ronen I. Kat, Elad Michael Schiller, Nir Tzachar, and Reuven Yagel. "Self-Stabilization from Theory to Practice". In: *Bulletin of the EATCS* 94 (2008), pp. 130–150.

[4] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. "Contention-free MAC protocols for asynchronous wireless sensor networks". In: *Distributed Computing* 21.1 (2008), pp. 23–42. DOI: 10.1007/s00446-007-0053-x. URL: http://dx.doi.org/10.1007/s00446-007-0053-x.

[5] Murat Demirbas, Onur Soysal, and Muzammil Hussain. "A Singlehop Collaborative Feedback Primitive for Wireless Sensor Networks". In: *INFOCOM 2008. 27th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 13-18 April 2008, Phoenix, AZ, USA*. IEEE, 2008, pp. 2047–2055. ISBN: 978-1-4244-2026-1. DOI: 10.1109/INFOCOM.2008.270. URL: http://dx.doi.org/10.1109/INFOCOM.2008.270.

[6] Manjunath Doddavenkatappa, Mun Choon Chan, and Akkihebbal L. Ananda. "Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed". In: *Testbeds and Research Infrastructure. Development of Networks and Communities - 7th International ICST Conference,TridentCom 2011, Shanghai, China, April 17-19, 2011, Revised Selected Papers*. 2011, pp. 302–316. DOI: 10.1007/978-3-642-29273-6_23. URL: http://dx.doi.org/10.1007/978-3-642-29273-6_23.

[7] Shlomi Dolev. *Self-Stabilization*. MIT Press, 2000. ISBN: 0-262-04178-2.

[8] Shlomi Dolev and Yinnon A. Haviv. "Self-Stabilizing Microprocessor: Analyzing and Overcoming Soft Errors". In: *IEEE Trans. Computers* 55.4 (2006), pp. 385–399. DOI: 10.1109/TC.2006.61. URL: http://dx.doi.org/10.1109/TC.2006.61.

[9] Shlomi Dolev, Yinnon A. Haviv, and Mooly Sagiv. "Self-stabilization preserving compiler". In: *ACM Trans. Program. Lang. Syst.* 31.6 (2009). DOI: 10.1145/1552309.1552312. URL: http://doi.acm.org/10.1145/1552309.1552312.

[10] Shlomi Dolev and Reuven Yagel. "Towards Self-Stabilizing Operating Systems". In: *IEEE Trans. Software Eng.* 34.4 (2008), pp. 564–576. DOI: 10.1109/TSE.2008.46. URL: http://dx.doi.org/10.1109/TSE.2008.46.

[11] Ted Herman and Sébastien Tixeuil. "A Distributed TDMA Slot Assignment Algorithm for Wireless Sensor Networks". In: *Algorithmic Aspects of Wireless Sensor Networks: First International Workshop, ALGOSENSORS 2004, Turku, Finland, July 16, 2004. Proceedings*. Vol. 3121. LNCS. Springer, 2004, pp. 45–58. ISBN: 3-540-22476-9. DOI: 10.1007/978-3-540-27820-7_6. URL: http://dx.doi.org/10.1007/978-3-540-27820-7_6.

[12] Arshad Jhumka and Sandeep S. Kulkarni. "On the Design of Mobility-Tolerant TDMA-Based Media Access Control (MAC) Protocol for Mobile Sensor Networks". In: *Distributed Computing and Internet Technology, 4th International Conference, ICDCIT 2007, Bangalore, India, December 17-20, Proceedings*. Ed. by Tomasz Janowski and Hrushikesha Mohanty. Vol. 4882. LNCS. Springer, 2007, pp. 42–53. ISBN: 978-3-540-77112-8. DOI: 10.1007/978-3-540-77115-9_4. URL: http://dx.doi.org/10.1007/978-3-540-77115-9_4.

[13] F. Kauer, M. Köstler, T. Lübkert, and V. Turau
. "OpenDSME - a portable framework for reliable wireless sensor and actuator networks". In: *2017 International Conference on Networked Systems, NetSys 2017, Göttingen, Germany, March 13-16, 2017*. 2017, pp. 1–2. DOI: 10.1109/NetSys.2017.7931495. URL: https://doi.org/10.1109/NetSys.2017.7931495.

[14] Florian Kauer, Maximilian Köstler, Tobias Lübkert, and Volker Turau. "Formal Analysis and Verification of the IEEE 802.15.4 DSME Slot Allocation". In: *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. Malta, Nov. 2016, pp. 140–147.

[15] Fabian Kuhn, Nancy A. Lynch, and Calvin C. Newport. "The abstract MAC layer". In: *Distributed Computing* 24.3-4 (2011), pp. 187–206. DOI: 10.1007/s00446-010-0118-0. URL: http://dx.doi.org/10.1007/s00446-010-0118-0.

[16] Christoph Lenzen, Jukka Suomela, and Roger Wattenhofer. "Local Algorithms: Self-stabilization on Speed". In: *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*. 2009, pp. 17–34. DOI: 10.1007/978-3-642-05118-0_2. URL: http://dx.doi.org/10.1007/978-3-642-05118-0_2.

[17] Florian Meier. "Ph.D. Forum Abstract: Scalable Wireless Networks for Industrial Control Systems with Time and Reliability Constraints". In: *Proceedings of the 15th ACM/IEEE Conference on Information Processing in Sensor Networks*. Vienna, Austria, Apr. 2016.

[18] Florian Meier and Volker Turau. "An Analytical Model for Fast and Verifiable Assessment of Large Scale Wireless Mesh Networks". In: *Proceedings of the Design of Reliable Communication Networks (DRCN)*. Kansas City, MO, USA, Mar. 2015, pp. 185–190.

[19] Thomas Petig, Elad Schiller, and Philippas Tsigas. "Self-stabilizing TDMA algorithms for wireless ad-hoc networks without external reference". In: *13th Annual Mediterranean Ad Hoc Networking Workshop, MED-HOC-NET 2014, Piran, Slovenia, June 2-4, 2014*. IEEE, 2014, pp. 87–94. DOI: 10.1109/MedHocNet.2014.6849109. URL: http://dx.doi.org/10.1109/MedHocNet.2014.6849109.

[20] Henning Tuan Hy Phan. "Towards Wireless Communication with Bounded Delay". MA thesis. Gothenburg, Sweden: Department of Computer science, Chalmers University of Technology, 2016.

[21] Injong Rhee, Ajit Warrier, Jeongki Min, and Lisong Xu. "DRAND: Distributed Randomized TDMA Scheduling for Wireless Ad Hoc Networks". In: *IEEE Trans. Mob. Comput.* 8.10 (2009), pp. 1384–1396. DOI: 10.1109/TMC.2009.59. URL: http://dx.doi.org/10.1109/TMC.2009.59.

[22] Raphael Rom and Fouad A. Tobagi. "Message-Based Priority Functions in Local Multiaccess Communication Systems". In: *Computer Networks* 5 (1981), pp. 273–286. DOI: 10.1016/0376-5075(81)90004-0. URL: http://dx.doi.org/10.1016/0376-5075(81)90004-0.

[23] Johannes Schneider and Roger Wattenhofer. "Coloring unstructured wireless multihop networks". In: *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*. 2009, pp. 210–219. DOI: 10.1145/1582716.1582751. URL: http://doi.acm.org/10.1145/1582716.1582751.

[24] Riccardo Scopigno and Hector Agustin Cozzetti. "GNSS Synchronization in Vanets". In: *NTMS 2009, 3rd International Conference on New Technologies, Mobility and Security, 20-23 December 2009, Cairo, Egypt*. 2009, pp. 1–5. DOI: 10.1109/NTMS.2009.5384821. URL: http://dx.doi.org/10.1109/NTMS.2009.5384821.

[25] Riccardo Scopigno and Hector Agustin Cozzetti. "Mobile Slotted Aloha for Vanets". In: *Proceedings of the 70th IEEE Vehicular Technology Conference, VTC Fall 2009, 20-23 September 2009, Anchorage, Alaska, USA*. IEEE, 2009, pp. 1–5. DOI: 10.1109/VETECF.2009.5378792. URL: http://dx.doi.org/10.1109/VETECF.2009.5378792.

[26] Dongjin Son, Bhaskar Krishnamachari, and John S. Heidemann. "Experimental study of concurrent transmission in wireless sensor networks". In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006, Boulder, Colorado, USA, October 31 - November 3, 2006*. 2006, pp. 237–250. DOI: 10.1145/1182807.1182831. URL: http://doi.acm.org/10.1145/1182807.1182831.

[27] Fan Yu and Subir K. Biswas. "Self-Configuring TDMA Protocols for Enhancing Vehicle Safety With DSRC Based Vehicle-to-Vehicle Communications". In: *IEEE Journal on Selected Areas in Communications* 25.8 (2007), pp. 1526–1537. DOI:

10.1109/JSAC.2007.071004. URL: http://dx.doi.org/10.1109/JSAC.2007.071004.

PAPER III

Shlomi Dolev, Thomas Petig and Elad Michael Schiller

Robust and Private Distributed Shared Atomic Memory in Message Passing Networks

Based on *Brief Announcement: Robust and Private Distributed Shared Atomic Memory in Message Passing Networks* that appeared in the proceedings of
2015 ACM Symposium on Principles of Distributed Computing (PODC)
Donostia-San Sebastián, Spain
July 21 - 23, 2015, pp. 311–313

4 Robust and Private Distributed Shared Atomic Memory

We study the problem of privately emulating shared memory in message passing networks. The system includes N servers and at most e semi-Byzantine servers that can deviate from the algorithm by sending corrupted data. Moreover, at most f servers can fail and stop.

The focus is on coded atomic storage (CAS) algorithms. We present a variant that ensures no information leakage by letting the servers store their data as secret shares. Our enhancement to CAS uses $\lceil (N + k + 2e)/2 \rceil$-size quorums and Reed-Solomon codes. This enhancement preserves the algorithm ability to function in asynchronous system settings.

To the best of our knowledge, we are the first to address the privacy issue when emulating shared memory in message-passing systems.

4.1 Introduction

Security and privacy are often imperative for distributed systems. This motivates us to study the problem of emulating shared memory in message passing networks that include N servers, at most f crash-stop failures and e semi-Byzantine servers that can deviate from the algorithm by sending corrupted data, but cannot deviate from the protocol. We look at coded atomic storage algorithms that ensure no information leakage by letting the servers store their data as secret shares. We consider an enhancement of the coded atomic storage (CAS) algorithm by Cadambe et al. [3] in which we use $\lceil (N + k + 2e)/2 \rceil$-size quorums and (N, k)-Reed-Solomon codes, where k represents the message length.

The first algorithm for emulating a single-writer multi-reader shared memory by Attiya et al. [1], as well as the multi-writer multi-reader version by Fan and Lynch [4], handle fail-stop failures and packet failures, such as packet omission, duplication and reordering. Spiegelman et al. [14] discuss the bounds on the space requirement under asynchrony. Furthermore, they provide an algorithm that is close to their lower bound. Cadambe et al. [3] present the coded atomic storage (CAS) algorithm and improve communication and storage costs by using quorums and (N, k)-maximum distance separable (MDS) codes [9]. The CAS algorithm enables the reader to restore the data under the presence of $\frac{N-k}{2}$ stop-failed servers. We address privacy by storing on each node merely parts of the data, as in Shamir's secret sharing scheme [10], which we can implement using Reed-Solomon codes [7] and a matching error correction algorithm (Berlekamp-Welch [15]). This variation of the CAS algorithm also provides resilience against other errors, for example, data corruption of a bounded number of secret shares. We show how to combine shared-memory emulation with robustness and privacy.

Background This short introduction in coding follows [9] and [5][1]. Shannon studied in 1948 communication over a noise channel and introduced a channel model [11]. Such a channel is given as $(\mathcal{X}, \mathcal{Y}, P_{Y|X})$, where $\mathcal{X}$ is the input alphabet, i.e., a set of symbols, $\mathcal{Y}$ is the output alphabet and the conditional probability $P_{Y|X}$ is the channel law. A special case is the discrete memoryless channel (DMC), which requires the channel law to fulfil $P_{Y_k|X_1,\ldots,X_k,Y_1,\ldots,Y_{k-1}}(y_k|x_1,\ldots,x_k,y_1,\ldots,y_{k-1}) = P_{Y|X}(y_k|x_k)$ for all k. This definition of the channel law implies in particular that the distribution of Y_k does not change over time. An example of a DMC is the binary erasure channel with erasure probability p_e. This channel is defined by $\mathcal{X} = \{0, 1\}$, $\mathcal{Y} = \{0, 1, \perp\}$, $P_{e|1} = P_{e|0} = p_e$ and $P_{1|1} = P_{0|0} = 1 - p_e$. In this channel, a symbol is erased with probability p_e, where earasure means that it is replaced by $\perp$.

Maximum Distance Separable Codes

To tackle these noisy channels and to increase the amount of information we can transmit over it we use coding schemes. A $(2^{\lceil NR \rceil}, N)$ coding scheme over the alphabet $\mathcal{X}$ is given by a set of messages $M := \{1, \ldots, 2^{\lceil NR \rceil}\}$, an encoder $\Phi : M \to \mathcal{X}^N$ and a decoder $\Psi : Y^N \to \hat{M} := M \cup \{\perp\}$, where $\perp$ is the error symbol for case the decoder fails. We

[1] The lecture notes of Stefan M. Moser, `http://moser-isi.ethz.ch/lectures.html`, are also worth to read.

say $\Phi(m)$ is a code word and the image of Φ is the codebook $\mathcal{C}$. The rate $0 < R \leq 1$ is a measure for the redundancy used, i.e., how much longer the code words are compared to the message. The block length N is the length of the code words.

For an alphabet A, we denote the Hamming distance of two words $x, y \in A^N$ by $d_{\text{Hamming}}(x, y) := \sum_{i=1}^{N} \mathbb{1}_{x_i \neq y_i}$. We say a $(|M|, N)$ coding scheme is a $(|M|, N, d)$ coding scheme if d is the minimum distance $d := \min_{\Phi(m_1) \neq \Phi(m_2)} d_{\text{Hamming}}(\Phi(m_1), \Phi(m_2))$. We denote by $\text{GF}(q)$ the Galois field of size $q \in \mathbb{N}$. A linear $|M|, N, d$ coding scheme over an alphabet $\mathcal{X} = \text{GF}(q)$ is a $|M|, N, d$ coding scheme, such that for all $m_1, m_2 \in M$ and all $x_1, x_2 \in \mathcal{X}$ exists $m_3 \in M$, such that $x_1\Phi(m_1) + x_2\Phi(m_2) = \Phi(m_3)$. Or, in other words, $\mathcal{C}$ is a linear subspace of $\mathcal{X}^N$ over $\mathcal{X}$. Thus, we can denote the dimension of $\mathcal{C}$ as k and we denote a linear $(|M|, N, d)$ coding scheme also as (k, N, d) coding scheme.

The Singleton bound for a $(|M|, N, d)$ coding scheme is given by $d \leq N - (\log_q(|M|) + 1$ and for a linear (k, N, d) coding scheme by $d \leq N - K + 1$ [12]. We call a coding scheme that attains equality maximum distance separable (MDS).

As above, let $M = \mathcal{X}^k = \text{GF}(q)$. Let $\alpha_1, \ldots, \alpha_{q-1} \in \mathcal{X}$ and non-zero. We us the Vandermonde matrix

$$G = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_{q-1} \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_{q-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_{q-1}^{k-1} \end{pmatrix}$$

as generator matrix. We define a Reed-Solomon coding scheme[2] as a mapping $\Phi(m) = mG$. Thus $\Phi(m)_i = \sum_{j=0}^{k-1} m_j \alpha_i^j$ and, due to the properties of a Galois field, $m_0 = -\sum_{i=1}^{q-1} \Phi(m)_i$. It is easy to see that this is a linear coding scheme. A proof that Reed-Solomon codes are maximum distance separable can be found in [9].

Secret Sharing

Let us look for a moment at secret sharing. The (N, k)-*threshold scheme* for integers k and N, such that $0 < k \leq N$, is defined by Shamir [10] and splits a secret s into N secret shares $\{s_i\}_{i \in \{1,\ldots,N\}}$. This scheme requires that there exists a mapping from any $S \subseteq \{s_i\}_{i \in \{1,\ldots,N\}}$ with $|S| \geq k$ to the secret s, but it is impossible to determine s from a set of less than k secret shares. Shamir gives an example for a (N, k)-*threshold scheme* using polynomials [10] on a Galois field $\text{GF}(q)$ for some prime q. We assume we can map our secret to some number, $m_0 \in \text{GF}(q)$ and we choose $m_1, \ldots, m_{k-1}$ randomly from a uniform distribution over $\text{GF}(p)$. Now we use the polynomial $Q(\alpha) := \sum_{j=0}^{k-1} m_j \alpha^j$ in $\text{GF}(p)$ to generate the secret shares $Q(1), Q(2), \ldots, Q(N)$. Given k of there secret shares we can interpolate the polynomial and generate the secret $m_0 = Q(0)$. By only having $k - 1$ secret shares, all p elements of $\text{GF}(p)$ are equally likely to be the secret.

The (N, k)-Reed-Solomon coding scheme can be used as a (N, k)-threshold scheme [7]. To implement this we let $\mathcal{X} = \text{GF}(p)$ for a prime p and we choose $\alpha_i = i$. We store the

[2] This generator matrix actually gives us only a special case of Reed-Solomon codes

Algorithm 2: Berlekamp-Welch decoder, Ψ as based on [6] for a N, k-Reed-Solomon coding scheme with $e < \frac{N-k+1}{2} - f$ errors.

input : Set $I = \{(i, w_i)\}_i$
output: Secret s

1.1 **if** $|\{w_i : (i, w_i) \in I, w_i \neq \perp\}| \geq k + 2e$ **then return** $\perp$;
1.2 Compute $E, P \in \mathrm{GF}(p)[X]$ with $\deg(E) = e$ and $\deg(P) = k - 1 + e$ and $E \neq 0$ such that $w_i E(i) = Q(i)$ for all $(i, w_i) \in I$ with $w_i \neq \perp$;
1.3 **if** *E does not divide Q* **then return** $\perp$;
1.4 Let $P := Q/E$;
1.5 **if** $|\{i : P(i) \neq w_i\}| > e$ **then return** $\perp$;
1.6 **return** P;

secret in m_0 and choose $m_1, \ldots, m_{k-1}$ randomly from a uniform distribution over $\mathrm{GF}(p)$. Then $\Phi(m)_i = \sum_{j=0}^{k-1} m_j \alpha_i^j = Q(i)$ and the Reed-Solomon coding scheme defined above corresponds to a (N, k)-threshold scheme with secret shares $\Phi(m)_i$.

Our output alphabet is $\mathcal{Y} = \mathrm{GF}(p) \cup \{\perp\}$. Meaning, we assume erasures which we denote by a special erasure symbol, $\perp$, as in the BEC and, thus, we know the position of the erasure. To reconstruct the secret, we need a decoder for the Reed-Solomon coding scheme. One decoder was presented in the paper by Reed and Solomon [8], but it is impractical since it requires $\mathcal{O}(\binom{N}{k})$ steps. The Berlekamp-Welch algorithm, Ψ, can correct (N, k)-Reed-Solomon codes within $\mathcal{O}(N^3)$ time in the presence of e errors and f erasures, as long as $2e + f < N - k + 1$ [15] as described in [6]. This algorithm is presented here as Algorithm 2. Line 1.2 can be solved in $\mathcal{O}(N^3)$ time using Gaussian elimination. Intuitively speaking, an erasure is not a problem as long as there are enough points left to interpolate the polynomial. Each symbol that is changed in the code word requires one additional as redundancy to allow us to detect the error and recover the polynomial and with this the message.

Our contribution We show how to emulate atomic shared memory in the presence of semi-Byzantine servers. This approach ensures privacy. Namely, no group of up to $k-1$ servers is able to reconstruct the stored data, i.e., the secret. Furthermore, a reader can reconstruct the correct secret even if up to e servers deliver corrupted secret shares. We do that using Reed-Solomon codes [8] and the Berlekamp-Welch error correction algorithm [15]. This works because Cadambe et al. [3] use the class of maximum-distance separable codes for their CAS algorithm, which includes the Reed-Solomon codes.

4.2 System Settings

We consider message passing networks in which nodes exchange messages via communication links. Messages are of the form $(t, w, d) \in \mathcal{T} \times \mathcal{W} \cup \{\perp\} \times \mathcal{D}$, where $\mathcal{T}$ is the set of tag tuples (z, i) that contain an integer z and a node identifier i. With $\mathcal{W}$ we denote the set of secret shares, where $\perp$ is the invalid share and $\mathcal{D} := \{\text{'pre'}, \text{'fin'}\}$ is the label set. We assume that $\mathcal{T}$ is lexicographical ordered. We distinguish among three node types: *server*, *reader* and *writer*. Each writer and each reader is reliably connected to all

Algorithm 3: The robust and private coded atomic storage algorithm, code for p_i. The difference to CAS [3] lays in the encoding during write operations, decoding during read operations and the different quorum size.

2.1 **Writer**: `// Writes secret` s.
2.2 Query for the highest finalised tag from a quorum, select the message $((z,k),w,\text{'fin'})$ such that z is max.;
2.3 pre-write: Send $((z+1,i),\Phi_{p_j}(s),\text{'pre'})$ to all $p_j \in \mathcal{P}$ and wait until quorum acknowledges;
2.4 finalise: Send $((z+1,i),\perp,\text{'fin'})$ to all $p_j \in \mathcal{P}$ and wait until quorum acknowledges;

2.5 **Reader**: `// Returns secret` s, `or` $\perp$ `in case of failure.`
2.6 Query for the highest finalised tag from a quorum, select the tag $((z,j),w_{p_j},\text{'fin'})$ such that z is maximal;
2.7 Finalise: Send $(t,\perp,\text{'fin'})$ to all $s \in \mathcal{P}$ and let Q be the set responses of a quorum of servers containing a tuple (j,w_j) for each server p_j that responded with a secret share $w_j \neq \perp$;
2.8 **return** $\Psi(Q)$

2.9 **Server**: Storage variable: $S \subset \mathcal{T} \times (\mathcal{W} \cup \{\perp\}) \times \{\text{'pre'},\text{'fin'}\}$;
2.10 **upon** (receive query) **do**
2.11 Reply with highest finalised tag;

2.12 **upon** (receive pre-write $(t,w,\text{'pre'})$) **do**
2.13 **if** $\nexists(t,\bullet) \in S$ **then** $S \leftarrow S \cup (t,w,\text{'pre'})$;
2.14 Reply with acknowledgement;

2.15 **upon** (receive finalise $(t,\perp,\text{'fin'})$ from writer) **do**
2.16 **if** $\exists(t,w,$ *'pre'*$) \in S$ **then**
2.17 $S \leftarrow (S \setminus \{(t,w,\text{'pre'})\}) \cup (t,w,\text{'fin'})$;
2.18 **else** Add $(t,\perp,\text{'fin'})$ to S;
2.19 Reply with acknowledgement and gossip $(t,\text{'fin'})$;

2.20 **upon** (receive finalise $(t,\perp,\text{'fin'})$ from reader) **do**
2.21 **if** $\exists(t,w,\bullet) \in S : w \neq \perp$ **then**
2.22 $S \leftarrow (S \setminus \{(t,w,\bullet)\}) \cup \{(t,w,\text{'fin'})\}$; reply (i,w) and gossip $(t,\text{'fin'})$;
2.23 **else** $S \leftarrow S \cup \{(t,\perp,\text{'fin'})\}$; reply $(i,\perp)$ and gossip $(t,\text{'fin'})$;

2.24 **upon** (receive gossip $(t,\perp,\text{'fin'})$ from server) **do**
2.25 **if** $m := \exists(t,\perp,\bullet) \in S$ **then** $S \leftarrow (S \setminus \{m\}) \cup (t,w,\text{'fin'})$;
2.26 **else** Add $(t,\perp,\text{'fin'})$ to S;

servers and all servers are connected with each other, as described in Cadambe et al. [3]. Let $\mathcal{P}$ be the *server* set, where $N := |\mathcal{P}|$. Each server has a local storage S for recieved messages.

The multi-writer, multi-reader (MWMR) shared memory emulation problem in message passing systems, is the problem of emulating a shared register in the above settings. This register can be atomically changed by the writers, and atomically read by the readers, while tolerating fail-stop failure of at most f server. The parameter k restricts the size of the part of the shared register that can be stored on a single server, i.e., each server can store up a fraction of $\lceil k^{-1} \rceil$ of the size of the register. A solution for $1 \leq k \leq N-2f$ is the CAS algorithm [3].

In addition to the requierments MWMR shared memory emulation problem, the robust and private distributed shared atomic memory problem allows server to deliver corrupted information and we require privacy. Our settings are motivated by (reliable) servers that

stores large secret shares on (unreliable) mass storage systems. We allow at most e semi-Byzantine servers and at most f failures. We assume that semi-Byzantine servers can send corrupted secret shares to readers, but not corrupted tags or labels, i.e., when a semi-Byzantine server replies with a tuple (t, w, d), only w might be corrupted.

In the proposed Algorithm 3, the writers split secrets using the (N, k)-Reed-Solomon code and submit the resulting secret shares to the servers. Servers store their secret shares and deliver them to the readers upon request. The proposed solution withstands a fault model that includes both server stop-failure and server semi-Byzantine behaviour. Note that this is an erasure channel, but, in contrast to the BEC, with memory. The probability of an erasure, i.e., a server stop-failed, depends on previous server stop-failures, because a stop-failed server stays stop-failed. The upper bounds on the number of stop-failed servers, as well as semi-Byzantine servers allows the Berlekamp-Welch Algorithm 2 to always be able to correct the errors and erasures and, thus, it never returns $\perp$ in our case.

We say that a secret sharing protocol is *t-private* when a set of at most t servers cannot compute the secret, as in [2]. Note that a 0-private protocol preserves no privacy. When the presence of at most t semi-Byzantine servers and at most s stop-failed servers does not influence the correctness of secret restored by a reader, we say that the protocol is *(s, t)-robust*, similar to *t-resilience* in [2].

Quorums of $(k + 2e)$-overlap Quorum systems can be used for ensuring transaction atomicity in replica system despite the presence of network failures [13]. We define a *quorum* as a server subset $Q \subseteq \mathcal{P}$ with at least $\lceil \frac{N+k+2e}{2} \rceil$ elements, and we write $\mathcal{Q}$ as the set of all quorums. Lemma 3 uses the quorum definition to show that any two different quorums share at least $k+2e$ servers, rather than just k of them as in Cadambe et al. [3]. These quorums guarantees that once a writer finishes its write operation, any reader can retrieve at least $k + 2e$ secret shares and reconstruct the secret. We have to show, similar to Cadambe et al. [3], that two different quorums share by definition at least $k + 2e$ server. This guarantees that after a writer wrote to a quorum, a reader can read enough values to reconstruct the secret.

Lemma 3. (Variation of [3], Lemma 5.1) *Suppose that $1 \leq k \leq N - 2f - 2e$. **(1)** If $Q_1, Q_2 \in \mathcal{Q}$, then $|Q_1 \cap Q_2| \geq k + 2e$. **(2)** The existence of such a k implies the existence of $Q \in \mathcal{Q}$ such that Q has no crashed servers.*

Proof. **(1)** Let $Q_1, Q_2 \in \mathcal{Q}$, then $|Q_1 \cap Q_2| = |Q_1| + |Q_2| - |Q_1 \cup Q_2| \geq 2 \left\lceil \frac{N+k+2e}{2} \right\rceil - N \geq k + 2e$. **(2)** Since there are at most f crashed servers, we can show that without such f servers, there are still enough alive servers for a quorum. It follows that $N - f \geq N - \left\lfloor \frac{N-k-2e}{2} \right\rfloor = \left\lceil \frac{N+k+2e}{2} \right\rceil$. □

By Lemma 3, the atomicity and liveness analysis in [3, Theorem 5.2 to Lemma 5.9] also holds when the CAS algorithm that uses $(k + 2e)$-overlap quorums.

4.3 The Algorithm

In order to tolerate at most e semi-Byzantine servers and the corrupted secret shares they send to a reader, we propose Algorithm 3 as a variation of Cadambe et al. [3] CAS algorithm that uses $(k+2e)$-overlap quorums and (N,k)-Reed-Solomon codes [8], which is an (N,k)-MDS [9] code that Cadambe et al. [3] uses. By the atomicity and liveness analysis for the case of $(k+2e)$-overlap quorums (the remark after Lemma 3), the reader retrieves $k+2e$ unique secret shares that include at most e manipulated shares. The server itself does not differ from [3], since it only stores the secret shares and it does not read or manipulate them. The meta data is identical to [3]. A secret is written by one of the writer using three phases: query, pre-write and finalise. First it queries for the maximal tag number, but it only waits for the replies from a quorum. This means it might not use the maximal tag number in the system, which might in the mean time increase anyway, due to concurrent write operations. The new tag tuple for this write operation consist of an incremented tag value and the node id of the current writer. In the pre-write phase the individual secret shares are generated for each server and delivered together with the new tag tuple. The last phase, finalise, makes the tags on the server visible for read operations since the pre-write waits for a quorum to acknowledge, it is ensured a read operation can read enough secret shares to decode the secret. After receiving the finalise message, a server sends a gossip messages to all other servers (Line 2.19).

A read operation consists of two phases. The query is identical to the write operation. Instead of incrementing, the tag tuple is used as it is for a finalise step. The finalise step asks for the secret share associated with this tag tuple. Since we use quorums, it is ensured that enough overlap exists and that the reader receives enough secret shares to decode the secret.

Corollary 1. *For $1 \leq k \leq N-2f-2e$, Algorithm 3 emulates a shared atomic read/write memory.*

Robustness Robustness is added by the ability of the Berlekamp-Welch algorithm to correct error in the Reed-Solomon codes. Note that semi-Byzantine servers only introduce corrupted secret shares. So, we do not need to handle corrupted tags, or labels. Lemma 4 shows the robustness of Algorithm 3 against up to e semi-Byzantine servers and up to f stop-failed servers.

Lemma 4. *For $k \in \{1\ldots, N-2f-2e\}$, Algorithm 3 (f,e)-robust.*

Proof. If a writer issues a query, pre-write and finalise operations it does not read back the secret from the server. Thus, writers are immune to semi-Byzantine servers. Servers do not exchange secrets with other servers and thus are not directly affected by semi-Byzantine servers. A reader collects secret shares from quorum of servers, but never writes them to servers, since a query and a finalise only contains a $\perp$ in place of a secret share. By Lemma 3 and Corollary 1 follows that a reader p_i receives at least $k+2e$ secret shares from the finalise operation. From these $k+2e$ secret shares at most e are corrupted and, thus, p_i computes the correct secret by applying Berlekamp-Welch. □

Privacy Our approach ensures privacy of the secret among servers. In Lemma 5 we see that a group of less than k servers are not able to reconstruct the secret by combining the secret shares they have stored locally.

Lemma 5. *For $1 \leq k \leq N - 2f - 2e$, Algorithm 3 is $(k-1)$-private.*

Proof. Let t be a tag and $k > 1$. A set of $k-1$ servers store together $k-1$ secret shares associated to the tag t. Since the secret shares encode a secret using Reed-Solomon codes, it is impossible to compute the original secret with less than k secret shares [7]. The case of $k = 1$ implies that the secret shares are the secret itself and, thus, privacy is compromised, i.e., it is 0-private. It follows that Algorithm 3 is $(k-1)$-private. □

Note that in the case $k = 1$, even if privacy is not protected, it is still possible to correct corrupted memory copies. This holds because the reader reads $1 + 2e$ secret shares and, thus, the additional $2e$ secret shares contain redundant information for the Berlekamp-Welch error correction.

4.4 Conclusions

Interestingly, fundamental building blocks for distributed systems can provide privacy and robustness. We show how to implement a robust and private coded atomic storage protocol, which is resilient to semi-Byzantine servers using shared memory emulation in message passing networks. In addition, our algorithm tolerates server crashes, and at the same time, it ensures the privacy of the stored data. We believe that our approach and techniques are useful for providing robustness and privacy for many more building blocks for distributed systems.

Bibliography

[1] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. "Sharing memory robustly in message-passing systems". In: *J. ACM (JACM)* 42.1 (1995), pp. 124–142.

[2] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *20th Symp. on Theory of Computing*. ACM, 1988, pp. 1–10. ISBN: 0-89791-264-0. DOI: 10.1145/62212.62213. URL: http://doi.acm.org/10.1145/62212.62213.

[3] Viveck R. Cadambe, Nancy A. Lynch, Muriel Médard, and Peter M. Musial. "A Coded Shared Atomic Memory Algorithm for Message Passing Architectures". In: *2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014, Cambridge, MA, USA, 21-23 August, 2014*. IEEE Computer Society, 2014, pp. 253–260. ISBN: 978-1-4799-5392-9. DOI: 10.1109/NCA.2014.44. URL: http://dx.doi.org/10.1109/NCA.2014.44.

[4] Rui Fan and Nancy A. Lynch. "Efficient Replication of Large Data Objects". In: *Distributed Computing, 17th International Conference*. Vol. 2848. LNCS. Springer, 2003, pp. 75–91. ISBN: 3-540-20184-X. DOI: 10.1007/978-3-540-39989-6_6. URL: http://dx.doi.org/10.1007/978-3-540-39989-6_6.

[5] Abbas El Gamal and Young-Han Kim. *Network Information Theory*. New York, NY, USA: Cambridge University Press, 2012. ISBN: 1107008735, 9781107008731.

[6] Peter Gemmell and Madhu Sudan. "Highly Resilient Correctors for Polynomials". In: *Inf. Process. Lett.* 43.4 (1992), pp. 169–174. DOI: 10.1016/0020-0190(92)90195-2. URL: http://dx.doi.org/10.1016/0020-0190(92)90195-2.

[7] R. J. McEliece and D. V. Sarwate. "On Sharing Secrets and Reed-Solomon Codes". In: *Commun. ACM* 24.9 (1981), pp. 583–584. ISSN: 0001-0782. DOI: 10.1145/358746.358762. URL: http://doi.acm.org/10.1145/358746.358762.

[8] Irving S Reed and Gustave Solomon. "Polynomial codes over certain finite fields". In: *J. Society for Industrial & Applied Math.* 8.2 (1960), pp. 300–304.

[9] Ron M. Roth. *Introduction to coding theory*. Cambridge Press, 2006. ISBN: 978-0-521-84504-5.

[10] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[11] C. E. Shannon. "A mathematical theory of communication". In: *The Bell System Technical Journal* 27.3 (July 1948), pp. 379–423. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb01338.x.

[12] R. Singleton. "Maximum Distance Q-nary Codes". In: *IEEE Transactions on Information Theory* 10.2 (Apr. 1964), pp. 116–118. ISSN: 0018-9448. DOI: 10.1109/TIT.1964.1053661.

[13] Dale Skeen. "A Quorum-Based Commit Protocol". In: *Berkeley Workshop*. 1982, pp. 69–80.

[14] Alexander Spiegelman, Yuval Cassuto, Gregory V. Chockler, and Idit Keidar. "Space Bounds for Reliable Storage: Fundamental Limits of Coding". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*. Ed. by George Giakkoupis. ACM, 2016, pp. 249–258. ISBN: 978-1-4503-3964-3. DOI: 10.1145/2933057.2933104. URL: http://doi.acm.org/10.1145/2933057.2933104.

[15] L.R. Welch and E.R. Berlekamp. *Error correction for algebraic block codes*. US Patent 4,633,470. 1986. URL: https://www.google.com/patents/US4633470.

PAPER IV

Thomas Petig and Elad M. Schiller and Jukka Suomela

Changing Lanes on a Highway

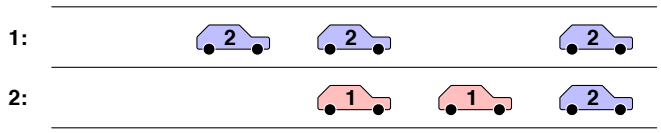
5 Changing Lanes on a Highway

We study a combinatorial optimisation problem that is motivated by the scenario of autonomous, collaborative agents driving on a multi-lane highway: some agents need to change lanes before the next intersection, and if there is congestion, the agents need to slow down to make space for those who are changing lanes. There are two natural objective functions to minimise: (1) the makespan, i.e., how long does it take for all traffic to clear the road, and (2) the total cost, i.e., the total number of manoeuvres over all agents. In this work we present efficient approximation algorithms for these problems in the two-lane case, and hardness results in the multi-lane case. We also discuss distributed, self-stabilising, and online versions of the problems.

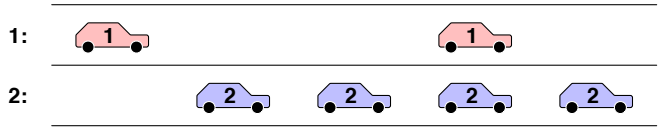
5.1 Introduction

5.1.1 Lane-Changing Problem

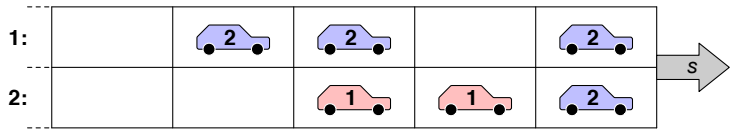
Consider a number of autonomous vehicles driving on a two-lane highway:



Each car is labelled with a lane number, 1 or 2, indicating where it needs to be before the next intersection. Our task is to instruct the cars to adjust their speed and change lanes so that all cars with label ℓ are on lane ℓ:

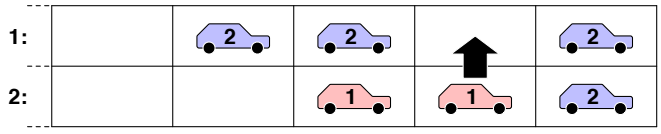


We discretise the traffic by assuming that there is a grid of *slots* that is moving at some fixed speed s (for example, s is the speed limit of the highway), and each car occupies one slot (there are infinitely many free slots behind the last cars):

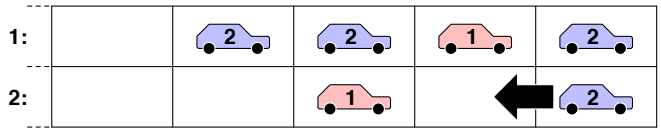


If there are no steering manoeuvres, each car will remain in its current slot (i.e., it is driving along the current lane, at a constant speed s). We can use the following manoeuvres to alter the relative positions of the cars.

First, a car that is *currently on the wrong lane* can ***switch lanes***, assuming there is an empty slot next to it:



Second, any car can ***slow down*** a bit to move backwards relative to the traffic around it, assuming there is an empty slot behind it:

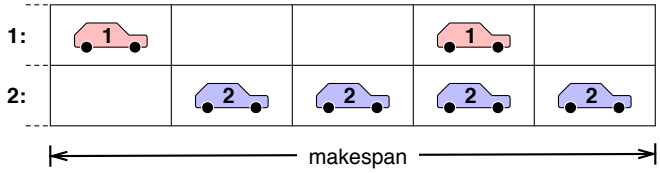


5.1.2 Objectives

It is easy to find a feasible solution by following a simple greedy strategy: for example, for each car x that is on the wrong lane, slow down all behind x on either lanes to make space for x to move to the right lane. However, this is clearly not an ideal strategy in the general case.

We will consider the following objective functions that we would like to minimise:

- **Makespan**: What is the last non-empty row that is occupied by a car in the final configuration? Intuitively, we measure here *how much do we stretch the traffic*, or equivalently, *how long does it take for all traffic to clear the road*.



- **Total cost**: What is the total number of steering manoeuvres (switching lanes or slowing down) that we need to solve the problem? Note that the number of lane changes is simply equal to the number of cars on the wrong lane, so the interesting question is the number of slow down operations. Intuitively, we measure here the *average delay for the traffic*.

In addition to these objectives, there is a third natural objective function that is interesting in this setting: **parallel time**, i.e., the number of time units needed to solve the problem, assuming that we can do multiple *non-conflicting steering manoeuvres* simultaneously in parallel. There are many reasonable definitions of non-conflicting manoeuvres (for example, can we have a consecutive chain of cars all slowing down simultaneously?), each of them requiring somewhat different strategies. We will leave a detailed discussion of minimising parallel time for future work, but we point out here that our algorithms have a structure that makes them easy to adapt them to a setting in which we can manipulate an entire chain of cars simultaneously in parallel.

5.1.3 Model of Computation

To focus on the most interesting algorithmic aspects, we present our algorithms from the perspective of a **global omniscient entity** that has a full control over all vehicles. However, we emphasise that our algorithms can be adapted to the following **distributed, online** setting:

- There is an infinite stream of cars driving along the highway.
- Lane changing is initiated when the first car reaches a fixed milestone (for example, a traffic sign warning about the intersection).

- The vehicles can send small messages to other vehicles in their immediate neighbourhood.

Furthermore, our algorithms can be made **self-stabilising**: we do not need to maintain any other logical state of the system beyond the current physical locations of the cars. We will discuss these aspects in more detail in the full version of this paper.

5.1.4 Contributions

In this work, we present polynomial-time algorithms for the **two-lane** version of the lane changing problem:

- a **1.5-approximation of the total cost** that is **makespan optimum**,
- an **exact algorithm** regarding the total cost.

We conjecture that there exists an efficient exact algorithm also for minimising the total cost. However, we show that the natural **multi-lane** extension of the problem is **NP-hard**. (The details of the hardness proof appears in Appendix 5.4.)

We note the resemblance between the problem studied by us in this work and combinatorial puzzles [21], such as the "15" Puzzle [9], which is a game that considers a framed four by four matrix that has 15 cell-tiles (agents); randomly ordered from 1 to 15. The goal of the game is to slide the tiles so that the tiles are ordered. For large-scale versions of the n-puzzle, finding an optimum solution is NP-hard [15, 16]. The problem studied by us considers a different kind of a combinatorial puzzle in which all the agents have one of ℓ target lanes assigned; many agents can share the same lane. Also, the "15" Puzzle [9] considers title sliding in all four directions whereas the studied problem considers agents that can either swap or delay, i.e., three directions rather than four. Moreover, the hardness result presented in this paper does not use techniques that are related to the ones that appear in [15, 16]. Feigenbaum et al. [4] formulated a number of graph problems for the semi-streaming model. Unlike their model, the studied problem does not allow a pair of nearby agents to swap cells, because the swap move requires the target cell to be empty.

Problems related to lane-change consider traffic streams from the point of view of vehicular control [14, 2, 7], traffic flow control [11], the scheduling of lane changes for autonomous vehicles [1], assessment of the situation before changing lane [17], and negotiation before lane changing [19] to name a few. It is often the case, as in [1, 14], that these problems consider a small set of nearby vehicles that need to coordinate a single lane-change manoeuvre. A number of recent efforts, such as the European project AutoNet2030 [19], considers the need to perform lane changes in congested traffic situations, as we do in this paper. Their study focuses on distributed mechanisms, i.e., the communication protocols, for enabling coordinated lane changes whereas this work focuses on (the difficulty of) finding an optimum solution with respect to the total number (and order) of vehicular manoeuvres. We view the solution proposed in this work as an essential component in the realisation of 25 years old vision of Automated Highway Systems (AHS) [18].

Cellular automata are often used for microscopic traffic flow prediction [12]. These models resembles the one of the studied problem in the sense that each vehicle occupies a single cell. However, Nagel [12] considers cellular automata that move the vehicles forward, whereas the studied models consider the vehicles that can merely change their current lanes or delay, i.e., the studied models consider silent moving forward actions because we do not aim at predicting all traffic patterns and just aim at minimizing the number of delays and lane changes. The systematic approach presented in [13] shows that their lane change rules can provide "realistic results" with respect to the system ability to offer an accurate traffic prediction. A complementary approach for studying the effect of lane-change behaviour via cellular automata [8] is the observation of driver behaviour [5, 22].

Wang et al. [20] offer a fully automated lane-changing controller that considers control variables for both (discrete) lane change times and (continuous) accelerations. Some of the vehicles make decisions to minimize the costs that are associated with undesirable situations and some vehicles minimize their own costs only. To the end of determining the controller behaviour, Wang et al. formulate a dynamic game problem, which they solve via an iterative numerical simulation at the microscopic level. The studied problem has different settings and optimization goals. Moreover, our analysis focuses on the structural aspects of the problem with respect to the computational complexity and approximation ratios.

Fang et al. [3] (and references therein) study a complementary problem of lane reservation that considers each a network of road segments that they call lanes and we call streams. There, each road segment has a limited capacity and thus vehicles reserve these road segments so that they travelling time is not added with unnecessary delay. Fang et al. use two integer linear programming models to formulate the problem and show that the complexity of the problem is non-deterministic polynomial-time hard.

We denote by $\mathbb{N} = \{1, 2, \ldots\}$ the set of natural numbers excluding 0. Let $A \subset \mathbb{N}$ be the set of agents[1]. For now, we consider the case of two lanes. We consider a set of targets $\mathcal{S} := \{1, 2\}$ and assume that the targets for the agents are given by a function $t : A \cup \{\bot\} \to \mathcal{S} \cup \{\bot\}$ with $\bot \mapsto \bot$ for the emptp space. For a vector $v := (v_1, \ldots, v_k) \in (A \cup \{\bot\})^k$ for some $k \in \mathbb{N}$ we define $t(v) := (t(v_1), \ldots, t(v_k))$. Let $j \in \{1, 2\}$ and $i \in \mathbb{N}$. We define an agent stream with 2 lanes[2], $R \in (A \cup \{\bot\})^{\mathbb{N}\times 2}$, as an infinite table with rows indexed by $\mathbb{N}$, and lanes indexed by $\{1, 2\}$ and each entry is either $\bot$, or an agent out of A. We require that every agent in A occurs exactly once in R. The $\bot$ symbol is a placeholder for empty space. We assume $R_{i,j} > R_{i',j'}$ if (i, j) is lexicographically larger then (i', j'). We define two (stream) operations, one for delaying to the next row and one for swapping between lanes, as following:

- **Delay at** (i, l)**,** $\downarrow(i, l)$**.** This operation delays the agent at $R_{i,l}$ to $R_{i+1,l}$ if $t(R_{i,l}, R_{i+1,l}) \in \mathcal{S} \times \{\bot\}$.
- **Swap at** (i)**,** $\leftrightarrow(i)$**.** This operation swaps one agent at $R_{i,1}$ or $R_{i,2}$ if $t(R_i) \in$

[1] This unique identifier could be related to the position in the input stream.
[2] The case of more than two lanes is discussed later in Section 5.4.

$\{(\perp, 1), (2, \perp)\}$.

Note that this requires that one of the swapped stream positions is $\perp$ and the other one an agent. We define a stream R to be feasible iff for all $l \in \{1, 2\}$ and for all $i \in \mathbb{N}$ we have $t(R_{i,l}) \in \{\perp, l\}$.

The *Agent Sorting* (AS) task given a stream R is to compute a feasible stream R' using legal slowing down and moving sideways operations. The *Optimum Agent Sorting* (OAS) task is to solve AS with the minimal sum of delay and swap operations.

For a lane $l \in \{1, 2\}$, we define $\bar{l} = 2$ if $l = 1$ and $\bar{l} = 1$ if $l = 2$.

Observation 1. *The exact amount of swaps needed is* $|\{R_{i,l} : t(R_{i,l}) \notin \{\perp, l\}\}|$.

We say an agent is coming from above into row i if it delays from row $i - 1$ to row i. In general, for a given row i, we refer to above in the stream, when we refer to rows j with $j < i$.

5.2 Upper Bounds

With Algorithm 4, we present an algorithm that computes an optimum solution for OAS for simple input streams and approximates an optimum solution for the general input. It consists of two for loops, where the first estimates the cost, i.e., the number of delays that are necessary to solve congestion, as well as, to make space for moving agents to their target lane and stores it for each row in d. The second loop iterates in reverse direction over the stream and moves the agents according to the computed cost. The output is then a feasible, but not necessary optimum, solution for OAS. We proof an approximation ratio of 1.5 later.

5.2.1 The Algorithm

The first loop ensures there is later enough free space to move the agents. From the congestion point of view, without loss of generality, let us look on lane 1. We have three cases for each row depending on how many agents in this row have the target lane 1:

1. If there are 2 agents, then the congestion increases by one.
2. If there is 1 agent, then the congestion does not change.
3. If there are 0 agents, then the congestion decreases by one.

The same three cases appear for lane 2. A special case occurs if $t(R_i) = (2, 1)$, where we need to ensure the d value is at least one and, therefore, there is space to delaying one and then swap both agents to their target lane.

Algorithm 4 implements this idea as follows. In line 4.6 that all cases of $(\perp, 1)$ and $(2, \perp)$ are resolved immediately. For the case $R_i = (2, 1)$, Algorithm 4 ensures d is at least 1 on both lanes in line 4.7 to generate space for swapping the agents at row i. In case there is a pair of agents with target 1 in lane i, $d_{i,1}$ is incremented in line 4.12.

Algorithm 4: OAS Approximation

```
4.1  Input: R;
4.2  Output: R;
4.3  variable: d = {0}^{|R|×2} ∈ ℕ^{|R|×2};
4.4  function ↓(i, l, a): for j ∈ ℕ from i to i + a − 1 do ↓(j, l);

4.5  for i ∈ ℕ from 1 to |R| do
4.6      if t(R_i) ∈ {(2, ⊥), (⊥, 1)} then ↔(i);
4.7      if t(R_i) = (2, 1) then d_i ← (max{1, d_{i,1}}, max{1, d_{i,2}}) ;
4.8      else
4.9          if t(R_{i,1}) ≠ 1 then d_{i,1} ← max{0, d_{i,1} − 1};
4.10         if t(R_{i,1}) = 2 then d_{i,2} ← d_{i,2} + 1;
4.11         if t(R_{i,2}) ≠ 2 then d_{i,2} ← max{0, d_{i,2} − 1};
4.12         if t(R_{i,2}) = 1 then d_{i,1} ← d_{i,1} + 1;
4.13     d_{i+1} ← d_i;
4.14 for i ∈ ℕ from |R| to 1 do
4.15     switch (t(R_{i,1}), t(R_{i,2})) do
4.16         case (1, ⊥) do ↓(i, 1, d_{i,1}); break;
4.17         case (⊥, 2) do ↓(i, 2, d_{i,2}); break;
4.18         case (1, 1) do ↓(i, 1, d_{i,1}); ↔(i); ↓(i, 1, d_{i,1} − 1); break;
4.19         case (2, 2) do ↓(i, 2, d_{i,2}); ↔(i); ↓(i, 2, d_{i,2} − 1); break;
4.20         case (2, 1) do ↓(i, 1); ↔(i + 1); ↔(i); ↓(i, 1, d_{i,1}); ↓(i + 1, 2, d_{i,2} − 1); break;
```

And the similar occurs for lane 2 in line 4.10. Any free space on one of the lanes leads to a decrement of $d_{i,1}$ in line 4.9 or of $d_{i,2}$ in line 4.11. This ensures d represents the congestion including the additional space needed to sort out $(2,1)$ cases.

The second loop performs the move operations starting from the back of the stream (line 4.14). The case of $(\bot,1)$ and $(2,\bot)$ was transformed in line 4.6 to $(1,\bot)$ and, respectively, to $(\bot,2)$. For the case of $t(R_i) = (1,\bot)$ in line 4.16 and for the case of $t(R_i) = (\bot,2)$ in line 4.17 the agent gets delayed according to the d value. The same for case of $t(R_i) = (1,1)$ in line 4.18 and for the case of $t(R_i) = (2,2)$ in line 4.19, but additional the agent, that is not on its target lane, is swapped to its target lane and delayed as well. The d value is sufficiently large for this, since it was incremented either in line 4.10, or in line 4.12. The last case, $t(R_i) = (2,1)$ is handled in line 4.20. First the agent on lane 1 is delayed one row. This is possible since we ensure that the d value is at least 1 in line 4.7. Afterwards, both agents are swapped to their target lane and then delayed according to the d values. By pushing the accumulated space requirements to the next rows (line 4.13), we ensure that the agents in row $i+1$ make enough space for the agents in row i.

5.2.2 The Analysis

In the following, we will denote the indicator function with $\mathbb{1}$. We denote with $|R|$ the largest index of an agent in R, i.e., $R_{|R|} \neq (\bot,\bot)$ and $R_j = (\bot,\bot)$ for all $j > |R|$.

Definition 1. Let R be an agent stream. An *l-congested region in R* is an integer interval $\{i, \ldots, i+a\}$ of maximal length, such that for every k with $1 \leq k \leq a+1$ we get $\text{cost}(i,k,l,R) := \sum_{j=i}^{i+k-1} \mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l} > k$. We call an agent stream R *simple* if there is no i such that $(t(R_{i,1}), t(R_{i,2})) \in \{(2,1), (\bot,1), (2,\bot)\}$.

We observe that the cost is upper bounded by $\text{cost}(i,k,l,R) \leq 2k$, since there are not more than two agents per row.

Lemma 6. *Let $1 \leq i_1 < i_2 < |R|$. Then* $\text{cost}(i_1, i_2 - i_1 + k, l, R) - \text{cost}(i_1, i_2 - i_1, l, R) = \text{cost}(i_2, k, l, R)$

Proof. Let $C := \text{cost}(i_1, i_2 - i_1 + k, l, R) - \text{cost}(i_1, i_2 - i_1, l, R)$.

$$\begin{aligned} C &= \sum_{j=i_1}^{i_1+i_2-i_1+k-1} \left(\mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l}\right) - \sum_{j=i_1}^{i_1+i_2-i_1-1} \left(\mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l}\right) \\ &= \sum_{j=i_2}^{i_2+k-1} \mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l} \\ &= \text{cost}(i_2, k, l, R). \end{aligned}$$

□

Lemma 7. *Let I_1 and I_2 be two different l-congested regions in R. Then $I_1 \cap I_2 = \emptyset$.*

Proof. Assume $I_1 \cap I_2 \neq \emptyset$. Let, $i_1 = \min I_1$, $i_1' = \min I_2$, $i_2 = \max I_1$ and $i_2' = \max I_2$. If $i_2 = i_2'$, then I_2 contradicts the maximal length requirement of an l-congested region, or $I_1 = I_2$. A similar argument holds for the case $i_1 = i_1'$. Without loss of generality, let $i_2 < i_2'$. But, then we can we can extend I_1 by adding the elements $i_2+1, i_2+2, \ldots, i_2' \in I_2$. By applying Lemma 6 every time, we see that it still fulfils the requirements of an l-congested region and therefore contradicts the maximality of I_1. Thus, if $I_1 \cap I_2 \neq \emptyset$ then $I_1 = I_2$. □

N.B. Lemma 7 does not hold if I_1 is a 1-congested region and I_2 is a 2-congested region. For an index i, with $1 \leq i \leq |R|$, exists maximal one 1-congested region and maximal one 2-congested region that include i.

Lemma 8. *Let R be an agent stream and let i with $1 \leq i \leq |R|$ be an index. The number of agents with target l that need to delay at least to row i in R is at least* $\text{cost}(i_0, i - i_0 + 1, l, R) - (i - i_0 + 1)$ *for every $i_0 < i$.*

Proof. The cost function $\text{cost}(i_0, i - i_0, l, R)$ returns the number of agents with target l between index i_0 and i. Since a feasible output can have maximal one agent per lane per row, we can leave $i - i_0$ agents between index i_0 and i. Therefore, $\text{cost}(i_0, i - i_0 + 1, l, R) - (i - i_0 + 1)$ represents the number of agents that cannot stay above index i in a feasible solution due to congestion. Thus, they need to be delayed even further. □

Lemma 9. *Let $I = \{i, i+1, \ldots, i+a\}$. The number of delays of agents with target l needed to convert the congested region I to a feasible stream is at least*

$$\sum_{k \in I} \left(\text{cost}(i, k-i+1, l, R) - (k-i+1)\right).$$

Proof. In Lemma 8 we saw a lower bound for the number of agents delaying over a given index. From Lemma 7 we know that there is no other congested region for this lane that intersects I. We now sum over all indexes that I covers. □

Lemma 10. *Algorithm 4 computes a feasible agent stream using a minimum number of swaps.*

Proof. To compute a feasible output, all cases of target pairs

$$(t(R_{i,1}), t(R_{i,2})) \in \{(\bot, 1), (2, \bot), (2, 1), (2, 2), (1, 1)\}$$

needs to be solved. The case $(\bot, 1)$ is converted to $(1, \bot)$, and the case $(2, \bot)$ is converted to $(\bot, 2)$, immediately with exactly one swap in row i in line 4.6. Line 4.7 ensures that both agents in the $(2, 1)$ case are delayed by at least one. This ensures that if agents are in the following row then they get delayed and there is space to swap the position of them in line 4.20.

We show that Algorithm 4 matches the lower bound with respect to swaps (Observation 1). Algorithm 4 is only doing swaps in line 4.18 and line 4.19 This does not include line 4.6, which we have already considered above. In both cases, the algorithm swaps an agent to its target lane. Thus, a necessary swap is performed. If the opposite lane is used by another agent in this row, the agent there is delayed at least once and therefore the swap is legal. This gives us the minimal number of swaps. Thus, Algorithm 4 matches the lower bound given in Observation 1.

Now, we show that delaying an agent at $R_{i,l}$ according to $d_{i,l}$ is legal. We point out that $d_{i,l} - 1 \leq d_{i+1,l} \leq d_{i,l} + 1$ for all i and for all l (cf. lines 4.7, 4.9, 4.10, 4.11 and 4.12). Let $i \in \mathbb{N}$ and $l \in \{1, 2\}$, such that $0 \leq i \leq |R|$, $t(R_{i,l}) = l$ and $d_{i,l} > 0$. We need to show that $R_{j,l} = \bot$ for all $j \in \{i+1, \ldots, i+d_{i,l}\}$, i.e., we can legally delay the agent at position (i, l) for $d_{i,l}$ times in the within the second for loop (line 4.14). We need to look at all agents in the interval $\{i+1, \ldots, i+d_{i,l}\}$ on both lanes.

Assume there is an agent at position $j \in \{i+1, \ldots, i+d_{i,l}\}$. If there is no such agent, we are done. In case $t(R_{j,l}) = \bar{l}$, i.e., the agent is not on its target lane in the input R, then this agent was swapped to its target lane in a previous iteration of this loop in the lines 4.18, 4.19, or 4.20. Thus, it is not in the way for a delay of the agent $R_{i,l}$ on lane l.

The proof also considers the cases of $t(R_{j,l}) = l$ and $t(R_{j,\bar{l}}) = l$. That is, there are agents with target lane l. We observe that $d_{j,l} > d_{i,l} - (j-i)$ and thus this agent is moved far enough in a previous iteration in this loop, since this loop goes from the largest row index to the smallest, we have that all agents placed in the input in rows with an index larger than i are moved before the agent at row index i. This happens for agents that are on their target lane in the lines 4.16 and 4.17, 4.18 and 4.19 as well as for agents not on their target lane in the lines 4.18, 4.19, and 4.20. □

Lemma 11. *Let R be a simple stream and $I := \{i_0, \ldots\}$ be an l-congested region in R. Let $i \in I$. Then $d_{i,l} = \max\{0, cost(i_0, i - i_0 + 1, l, R) - i + i_0 - 1\}$ at line 4.14 of Algorithm 4.*

Proof. Let i be an index $1 \leq i < |R|$ and l be lane. We see that $d_{i+1,l} \in \{d_{i,l} - 1, d_{i,l}, d_{i,l} + 1\}$. Especially, $d_{i+1,l} = d_{i,l} + \mathbb{1}_{t(R_{i+1,1})=l} + \mathbb{1}_{t(R_{i+1,2})=l} - 1$. Let $d_{i,l} > 0$ and let $i_1 \leq i$ be minimal such that $d_{j,l} > 0$ for all $j \in \{i_1, \ldots, i\}$. Thus, either $i_1 = 1$, or $d_{i_1-1,l} = 0$. In both cases, are two statements following. First, $d_{i_1,l} = 1$, because it can only increment by one. Second, $t(R_{i_1}) = (l, l)$, because there must be congestion to increment by one.

Furthermore, we see that

$$\begin{aligned} d_{i,l} &= d_{i-1,l} + \mathbb{1}_{t(R_{i,1})=l} + \mathbb{1}_{t(R_{i,2})=l} - 1 \\ &= \sum_{j=i_1}^{i} \mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l} - 1 \\ &= \left(\sum_{j=i_1}^{i} \mathbb{1}_{t(R_{j,1})=l} + \mathbb{1}_{t(R_{j,2})=l} \right) - (i - i_1 + 1) \\ &= \text{cost}(i, i - i_1 + 1, l, R) - (i - i_1 + 1). \end{aligned} \tag{5.1}$$

From (5.1) and the definition of i_1, it follows that i_1 is part of the l-congested region $I := \{i_0, \ldots\}$.

To conclude this lemma, we need to prove that $i_1 = i_0$, i.e., both i_0 and i_1 start the same l-congested region I. Assume that $i_0 < i_1$ (because we have just showed that $i_0 > i_1$ is not possible). This means that $t(R_{i_0,l}) = (l, l)$ and, therefore $\text{cost}(i_0, 0, l, R) = 1 = d(i_0, l)$. From Definition 1 it follows that for all $j \in I := \{i_0, \ldots, i\}$, we have $\text{cost}(i_0, j - i_0, l, R) > j - i_0$. And by using (5.1) we get $d_{j,l} > 0$. But, this is a contradiction to the minimality of i_1. □

Lemma 12. *Algorithm 4 solves OAS for simple agent streams.*

Proof. From Lemma 10, we know that Algorithm 4 computes a feasible agent stream using a minimal number of swaps. Since the input is simple, the condition in line 4.7 is always false. Now, let I be a l-congested region. We point out that Algorithm 4 delays an agent with target l at position i, l exactly $d_{i,l}$ times (lines 4.16, 4.17, 4.18, 4.19) and an agent with target l at position $i, \bar{l}$ exactly $d_{i,l} - 1$ times (lines 4.18 and 4.19).

From Lemma 11 we know that $d_{i,l} = \max\{0, cost(i_0, i - i_0 + 1, l, R) - i + i_0 - 1\}$ and from Lemma 8 we know that this is the number of agents that pass index i.

We need to show that $\sum_{k \in I} \mathbb{1}_{t(R_{k,l})=l} d_{i,l} + \mathbb{1}_{t(R_{k,\bar{l}})=l}(d_{i,l} - 1)$ is equal to the lower bound

from Lemma 9. For this we rearrange the sum:

$$\sum_{k\in I} \mathbb{1}_{t(R_{k,l})=l} d_{i,l} + \mathbb{1}_{t(R_{k,\bar{l}})=l}(d_{i,l}-1)$$

$$= \sum_{k\in I}\left(\sum_{i:i_0<i\leq k, t(R_{i,l})=l} \mathbb{1}_{i+d_{i,l}>k} + \sum_{i:i_0<i\leq k, t(R_{i,\bar{l}})=l} \mathbb{1}_{i+d_{i,l}-1>k} \right) \tag{5.2}$$

$$= \sum_{k\in I} \big(\mathrm{cost}(i, k-i+1, l, R) - (k-i+1)\big). \tag{5.3}$$

For (5.2) we sum over all row indices k the agents that cross this index and for (5.3) we use the assumption that I is a congested region, otherwise we get negative terms. We also use the assumption that the input is simple, so the $d_{i,l}$ only represent congestion, as it is counted by the *cost* function. Thus, the lower bound of Lemma 9 is met and, therefore, the number of delays is minimum. □

Definition 2. The OAS problem with the additional constraint that for every $t(R_i) = (2,1)$, neither the agent at $R_{i,1}$, nor at $R_{i,2}$, are allowed to stay at row i in the solution is called constrained optimum agent sorting (cOAS).

Lemma 13. *Algorithm 4 computes an optimum solution for cOAS.*

Proof. For every row i with $t(R_i) = (2,1)$ and, both, $d_{i,1} > 0$ and $d_{i,2} > 0$, we have that every agent at R_{i+1} is delayed due to the d_{i+1} values. This means there is space to sort the agents at row i without additional delays to the ones needed to resolve congestion.

The interesting case is if at least one of $d_{i,1}$ and $d_{i,2}$ is zero. First, we consider that both are zero. For the case of $d_{i,1} = 0$ and $d_{i,2} = 0$, we will need at least two additional delays compared to an optimum solution for OAS, one for each lane. We also know there is no agent moving from $R_{i-1,l}$ to $R_{i,l}$ for all l. Algorithm 4 is clearing row $i+1$, delaying a_2, swapping both agents and delaying a_1 in line 4.20. For now, take an optimum solution where a_1 ends on index i_1. Then, we can keep a_1 in place do all operations below $i+1$ and delay the whole block $R_{i+1,1}, \ldots, R_{i_1,1}$ and finally move a_1 to row $i+1$. This holds for a_2 as well. Thus, the solution of Algorithm 4 is for this case optimum with regard to cOAS.

For the other two cases, i.e., exactly one of $d_{i,1}$ and $d_{i,2}$ is larger than zero, we now assume, without loss of generality, that $d_{i,1} = 0$ and $d_{i,2} > 0$ and i is maximal. This means, agent $a_2 := R_{i,1}$, which has target $t(a_2) = 2$, needs to delay anyway due to congestion, as we have the assumption that $d_{i,2} > 0$. For $a_1 := R_{i,2}$ we need to add at least one additional delay to fulfil the constraints of cOAS. We know that only on lane 2 agents delay from $R_{i-1,2}$ to $R_{i,2}$, because $d_{i,1} = 0$. To conclude this lemma, we let S' be an optimum solution cOAS. In case a_1 does not end up at $(i+1,1)$, but at some $i_1 > i+1$ in S', one can rearrange the operation. Instead of the operations leading to S', one can apply all operations necessary to empty the position $(i_1,1)$, but none of the operations that move a_1. This means a_1 stays in position $(i_1,1)$. Then we delay the block from $(i+1,1), \ldots, (i_1-1,1)$ and delay a_1 to $(i+1,1)$ using the same amount

row	i	i+1		i_3
lane 1	a_2	$R_{i+1,2}$	...	$R_{i_3,2}$
lane 2	a_1	•	...	•

Figure 5.1: An initial setup in R.

row	i	i+1		i_3	i_3+1		i_2
lane 1	•		...			...	a_1
lane 2	a_2	$R_{i+1,2}$	...	$R_{i_3,2}$	b_1	...	

Figure 5.2: A solution S.

of delay operations. Since only twos coming from above, a_2 can swap in between, or in front of them for the same cost, which is based on congestion. Since only agents $a_{j,l} := R_{j,l} : j < i \wedge l \in \{1,2\}$ with lane $2 = t(a_j)$ as their target lane, move in S' from row $i-1$ to i. Agent a_2 can swap in between (or in front as well as after) the agents $a_{j,l}$. for the same cost. Thus, the output of Algorithm 4 has the same cost as any optimum solution cOAS. □

Lemma 14. *An optimum solution for cOAS does not use more than* 2 *times the delays of an optimum solution for OAS and* 1.5 *times the total amount of operations.*

Proof. Let R be an agent stream. Let S be an optimum solution of OAS on R. Let i be a minimal row index, such that $t(R_i) = (2,1)$ and, either $S_{i,1} = R_{i,2}$, or $S_{i,2} = R_{i,1}$. If there is no such i, then Lemma 14 follows immediatly, because S is also an optimum solution for cOAS.

Without loss of generality, we assume $S_{i,1} = R_{i,2} =: a_2$. Let $a_1 = R_{i,1}$ be the agent with $t(a_1) = 2$. According to our assumption, a_1 must delay at least once, since its neighbour a_2 swaps in row i. Let us assume it swaps lanes at row i_1 with $i_1 > i$, and its final position in S is $(i_2, 2)$ with $i_2 \geq i_1$.

Let $i_3 < i_1$ be maximum, such that for all $j \in \{i+1, \ldots, i_3\} =: \gamma$ we have $S_{j,1} = R_{j,2} \neq \perp$. This setup is shown in Figure 5.1. We denote with G the set of these agents in $\{R_{j,2}\}_{j\in\gamma}$. Assume there is an agent $b_1 = S_{i_3+1,1}$, as we show in Figure 5.2. Since γ is maximal, $b_1 \neq R_{i_3+1,2}$. The agent b_1 either followed after a_1 on lane 1 to its position in S, or it delayed on lane 2 after the agents in G swapped to lane 1. Thus, b_1 position in R was above row i.

If we look in the first case, we note that there cannot follow another after b_1 since the

row	i	i+1		i_3	i_3+1		i_2
lane 1	•		...			...	a_1
lane 2	b_1	$R_{i+1,2}$	...	$R_{i_3,2}$	a_2	...	

Figure 5.3: The modified solution S, such that a_2 does not stay on row i.

row	i	i+1		i_3	i_3+1		i_2
lane 1	•		...			...	a_1
lane 2	b_1	a_2	$R_{i+1,2}$	...	$R_{i_3,2}$	...	

Figure 5.4: The modified solution S, such that a_2 and $R_{i+1,2},\ldots,R_{i_3,2}$ delay by one.

agents in G only swap once and do not do any further operation and these agents block the way, i.e., every further delay is blocked by b_1, every swap blocked by an agent in G. In this case, one could delay a_2 instead of b_1 and leave b_1 at position $(i,1)$ and a_2 at position $(i_3,1)$ and a_2 fulfils the constraint for cOAS at the same cost and we obtain the solution presented in Figure 5.3.

For the second case, one could delay everyone in G and a_2 by one after they swapped. Note that this swap is necessary for b_1 to pass on on lane 2. Here b_1 stays at index i and swaps. Since a_2 delayed by 1 the constraint of cOAS is fulfilled without extra cost. We see this solution in Figure 5.4.

For the case that $S_{i_3+1,1} = \perp$, we delay all agents in G by one and then delay a_2. Thus, we fulfil the constraint of cOAS. We point out that a_1 delayed at least $|G|+1$ times and the agents in G perform $|G|$ swaps. Thus, we additionally pay at most $|G|+1$ delays. Therefore, we increase the number of delays by at at most a factor of 2 and we increase the number of total operations by at most a factor 1.5. □

Theorem 2. *Algorithm 4 is an* 1.5*-approximation for OAS.*

Proof. Apply Lemma 14 to Lemma 13. □

Definition 3. Makespan is defined as $\max\{i : t(R_i) \neq (\perp,\perp)\} - \min\{i : t(R_i) \neq (\perp,\perp)\}$.

Theorem 3. *Algorithm 4 computes the minimum makespan.*

Proof. Using the argumentation of the proof of Lemma 14, we see that the constructed solution by modifying the OAS solution S has the same makespan. But, this is not less than the makespan Algorithm 4 produces. Since after adding additional delays if necessary for the case the targets are $(2,1)$ in line 4.7, all further rows are sorted using a minimum number of operations as we saw in Lemma 13. □

5.3 Optimum Solution

The direct solver presented in Algorithm 5 has, as the approximation Algorithm 4, two stages. First, the cost is estimated, then the agents are moved. The main difference is that the queues in Algorithm 5 give us more details on how to move agents than the d values in Algorithm 4. This allows us to compute the minimum amount of operations that are necessary to compute a feasible solution for OAS. Nevertheless, we will see that for simple inputs, the length of the queues corresponds to the d values of the approximation Algorithm.

We use queues to store the order in which agents delay from on row to the next. This means if a queue $q_1^{(i)}$ contains the agents a_1, a_2 and a_3 in this order and where a_1 is the first, then to create the solution S and some point a_1 will delay from position $(i-1,1)$ to $(i,1)$ and then a_2 and afterwards a_3. Before and between these delays might other steps be executed. But, there will be no delay from $(i-1,1)$ to $(i,1)$ after a_3 delayed.

Algorithm 5 consists of two phases. First, for each row of the input it uses Algorithm 6 to fill up the queues for the next row. For this, it only uses the knowledge from above and it do not look ahead. The $|R|+1$'s call of line 5.2 is using an empty row. This allows Algorithm 6 to sort both queues, $q_{|R|,1}$ and $q_{|R|,2}$. Afterwards, only empty rows are following, so we just leave the agents in the order they appear in the queues in line 5.3. This means the last elements of $q_1^{(|R|+2)}$ and $q_2^{(|R|+2)}$ are stored in $S_{|R|+2}$, the second last in $S_{|R|+3}$ and so on. Afterwards we know already how the solution, S, looks like. To actually move the agents Algorithm 5 calls Algorithm 9 in line 5.4.

Algorithm 5: Directly Computing the optimum solution w/o wrong-side-moves in one pass.

input : Stream $R \in (I \cup \{\bot\})^{2\times|R|}$
output: Stream $S \in (I \cup \{\bot\})^{2\times|R|}$
`/* Assume that` $(q_1^{(i)}, q_2^{(i)}) = (\emptyset,\emptyset)$ `for all` $i \in \mathbb{N}$ `*/`
5.1 **for** $i \in \{1,\dots|R|+1\}$ **do**
5.2 $\quad (q_1^{(i+1)}, q_2^{(i+1)}, S_i) \leftarrow \text{SolveRow}(q_1^{(i)}, q_2^{(i)}, R_i)$;
5.3 $S_{|R|+2} \leftarrow (q_1^{(|R|+2)}, q_2^{(|R|+2)})$;
5.4 Apply moves from $(q_l^{(i)})_{1\le i\le|R|+2, l\in\{1,2\}}$ to R;

5.3.1 Correctness of the direct solver

Theorem 4. *Algorithm 5 terminates eventually.*

Proof. Algorithm 8 terminates in $O(1)$, since the *push* operation to a queue terminates in $O(1)$. Algorithm 7 iterates over all over all elements in both input queues and the 2 agents in the given row. For each iteration, it calls Algorithm 8. Thus, Algorithm 7 terminates within $O(|q_1|+|q_2|)$. Algorithm 6 iterates at most once over one of the input queues, to determine if the queue needs to be split before calling Algorithm 7. This has a time requirement of $O(\max\{|q_1|+|q_2|\})$. It splits a queue if necessary, calls Algorithm 7, and copies the rest of the queue at most once. This leads to an overall time requirement of $O(|q_1|+|q_2|)$. Note that every agent occurs at most once in the queues. Thus, we can rewrite the time requirement as $O(|R|)$.

The sort step in line 5.2 is executed $|R|+1$ times and essentially sorts the queues together with the agents on the current row. There cannot be more than $2i$ agents in the queues $q_1^{(i)}$ and $q_2^{(i)}$ together. We know Algorithm 6 requires $O(|R|)$. This leads to a time requirement of $O(|R|^2)$ for the first $|R|+1$ rows. line 5.3 takes at most $O(|R|)$, because we are just coping element wise.

Algorithm 6: SolveRow(): Directly Computing the optimum solution w/o wrong-side-moves for a given row.

input : Queues q_1 and q_2 and a row (r_1, r_2)
output: q_1, q_2, r_1, r_2

6.1 **if** $|q_1| = |q_2| = 0$ **then** SortStep$((r1, r2), q_1, q_2)$; $(l_1, l_2) \leftarrow (\text{pop}(q_1), \text{pop}(q_2))$;
6.2 **else if** $|q_1| = |q_2| > 0$ **then**
6.3 **if** $t(last(q_1)) = 1 \wedge t(last(q_2)) = 2$ **then**
6.4 $(l_1, l_2) \leftarrow (pop(q_1), pop(q_2))$ and Sort(q_1, q_2, r_1, r_2);
6.5 **else if** $t(last(q_2)) = 2$ **then**
6.6 $(l_1, l_2) \leftarrow (\perp, pop(q_2))$; Let j be max. s. t. $t(q_2(j)) = 1$, or -1 otherwise;
6.7 **if** $j \neq -1$ **then**
6.8 $l_1 \leftarrow q[j]$ Sort $q_1, q_2[0:j], r_1, r_2$ to (q_1', q_2'); Append $q_2[j+1:]$ to q_2';
6.9 $(q_1, q_2) \leftarrow (q_1', q_2')$;
6.10 **else if** $t(r_2) = 1$ **then** If $t(r_1) \neq \perp$ then prepend r_1 to q_1; $l_1 \leftarrow r_2$;
6.11 **else**
6.12 **if** $t(last(q_1)) = 1$ **then** $l_1 \leftarrow pop(q_1)$; ;
6.13 **else if** $|q_1| = 0 \wedge t(r_1) = 1$ **then** $l_1 \leftarrow r_1$; $r_1 \leftarrow \perp$;
6.14 Sort(q_1, q_2, r_1, r_2);
6.15 **else if** $t(last(q_1)) = 1$ **then** Same as for $t(last(q_2)) = 2$ with changed lanes ;
6.16 **else** $(l_1, l_2) = (pop(q_2), \perp)$; Sort q_1, q_2, r_1, r_2 to q_1, q_2 ;
6.17 **else if** $|q_1| < |q_2|$ **then**
6.18 **if** $t(last(q_2)) = 2$ **then**
6.19 $(l_1, l_2) \leftarrow (\perp, pop(q_2))$; Let j be max. s. t. $t(q_2(j)) = 1$, or -1 otherwise;
6.20 **if** $j \neq -1$ **then**
6.21 **if** $|q_1| = 1$ *and* $1 = t(q_1[0])$ *and* $q_1[0] < q_2[j]$ **then**
6.22 $l_1 \leftarrow pop(q_1)$ and Sort q_1, q_2, r_1, r_2 to (q_1, q_2);
6.23 **else**
6.24 Sort $q_1, q_2[0:j], r_1, r_2$ to (q_1', q_2'); Append $q_2[j+1:]$ to q_2';
6.25 $(l_1, q_1, q_2) \leftarrow (q_2(j), q_1', q_2')$;
6.26 **else if** $t(r_2) = 1$ **then** If $r_1 \neq \perp$ then prepend r_1 to q_1; $l_1 \leftarrow r_2$;
6.27 **else**
6.28 **if** $t(last(q_1)) = 1$ **then** $l_2 \leftarrow pop(q_1)$; ;
6.29 **else if** $|q_1| = 0 \wedge t(r_1) = 1$ **then** $l_1, l_2 \leftarrow r_1$; $r_1 \leftarrow \perp$;
6.30 Sort(q_1, q_2, r_1, r_2);
6.31 **else**
6.32 **if** $|q_2| > 0$ *and* $t(last(q_2)) = 2$ **then**
6.33 $(l_1, l_2) \leftarrow (\perp, pop(q_2))$; Let j be max. s. t. $t(q_2(j)) = 1$, or -1 otherwise;
6.34 **if** $j \neq -1$ **then**
6.35 **if** $|q_1| = 1$ *and* $1 = t(q_1[0])$ *and* $q_1[0] < q_2[j]$ **then**
6.36 $l_1 \leftarrow pop(q_1)$ and Sort q_1, q_2, r_1, r_2 to (q_1, q_2);
6.37 **else**
6.38 Sort $q_1, q_2[0:j], r_1, r_2$ to (q_1', q_2'); Append $q_2[j+1:]$ to q_2';
6.39 $(l_1, q_1, q_2) \leftarrow (q_2(j), q_1', q_2')$;
6.40 **else if** $t(r_2) = 1$ **then** If $r_1 \neq \perp$ then prepend r_1 to q_1; $l_1 \leftarrow r_2$;
6.41 **else** $(l_1, l_2) \leftarrow (pop(q_2), \perp)$; Sort q_1, q_2, r_1, r_2 to q_1, q_2. ;
6.42 **else** $(l_1, l_2) \leftarrow (pop(q_2), \perp)$; Sort q_1, q_2, r_1, r_2 to q_1, q_2. ;
6.43 **else** Same as for $|q_1| < |q_2|$, but with changed lanes ;
6.44 **return** q_1, q_2, l_1, l_2;

Algorithm 7: Sort():

input : Queues q_1 and q_2 and a row (r_1, r_2)
output: q_1, q_2

7.1 $(q_1', q_2') \leftarrow ([], [])$;
7.2 **while** $|q_1| + |q_2|$ **do**
7.3 $\quad$ *SortStep*(r_1, r_2, q_1', q_2');
7.4 $\quad$ **if** *first*$(q_1) >$ *first*(q_2) **then** $(r_1, r_2) \leftarrow$ (*unshift*$(q_1), \perp)$;
7.5 $\quad$ **else** $(r_1, r_2) \leftarrow (\perp,$ *unshift*$(q_2))$;
7.6 *SortStep*(r_1, r_2, q_1', q_2');
7.7 **return** q_1', q_2';

Algorithm 8: SortStep():

input : Queues q_1 and q_2 and a row (r_1, r_2)
output: q_1, q_2

8.1 **switch** r_1, r_2 **do**
8.2 $\quad$ **case** $(\perp, 1)$ **do** *push*(q_1, r_2);
8.3 $\quad$ **case** $(\perp, 2)$ **do** *push*(q_2, r_2);
8.4 $\quad$ **case** $(1, \perp)$ **do** *push*(q_1, r_1);
8.5 $\quad$ **case** $(1,\ 1)$ **do** *push*(q_1, r_1); *push*(q_1, r_2);
8.6 $\quad$ **case** $(1,\ 2)$ **do** *push*(q_1, r_1); *push*(q_2, r_2);
8.7 $\quad$ **case** $(2, \perp)$ **do** *push*(q_2, r_2);
8.8 $\quad$ **case** $(2,\ 1)$ **do** *push*(q_1, r_1); *push*(q_1, r_2);
8.9 $\quad$ **case** $(2,\ 2)$ **do** *push*(q_2, r_1); *push*(q_2, r_2);

Algorithm 9: Apply moves: Constructing a sequence of moves given solution of Algorithm 5

input : Queues $q_l^{(i)}$ and streams R and S.
output: For now we directly solve the stream, so the output is actually the same stream as the output S of Algorithm 5. Technically it is the sequence of moves we are doing to get from R to S.

9.1 **while** $|\{R_{i,l} : t(R_{i,l}) \notin \{\perp, l\}\}| > 0$ **do**
9.2 $\quad$ Select i, l such that $R_{i,l} = \perp$ and either *first*$(q_l^{(i)}) = R_{i-1,l}$, or $t(R_{i,\bar{l}}) = l$;
9.3 $\quad$ **if** *first*$(q_l^{(i)}) = R_{i-1,l}$ **then**
9.4 $\quad\quad$ **if** $(S_{i,l} = R_{i-1,l}$ *and* $|q_l^{(i+1)}| = 0)$ *or first*$(q_l^{(i+1)}) = R_{i-1,l}$ *or* $R_{i-1,l} \in q_{\bar{l}}^{(i+1)} \wedge R_{i-1,\bar{l}} = \perp$
$\quad\quad\quad$ *or* $S_{i,\bar{l}} = R_{i-1,l} \wedge 0 = |q_{\bar{l}}^{(i)} \cap \{R_{i,\bar{l}}\} \cup q_l^{(i+1)}|$ **then**
9.5 $\quad\quad\quad \downarrow(i-1, l)$ and *unshift*$(q_l^{(i)})$;
9.6 $\quad$ **else if** $R_{i,\bar{l}} = S_{i,l}$ *or first*$(q_l^{(i+1)}) = R_{i,\bar{l}}$ **then**
9.7 $\quad\quad \leftrightarrow(i)$;

Algorithm 9 is called in line 5.4. This Algorithm moves one agent at each iteration. The agents are sorted into queues, such that either their order is kept among the agents that stay on the same lane, as in lines 8.4, 8.3 and 8.6, or their is space for a swap, as in lines 8.5, 8.7, 8.8 and 8.9 and the fact Algorithm 8 is called in lines 7.3 and 7.6 only on the first elements of the queue, which is removed and then Algorithm 8 is called again on the new first element until there are no elements left.

We must delay at least one agent in R_i, if $t(R_i) = (2, 1)$. But, we can sort all the agents delaying from row $i-1$ to row i, by only delaying agents in one row per iteration, as we see in lines 7.4 and 7.5, while prioritising on the position in R to ensure the feasibility of this delay.

This ensures that if there is an agent in the way, meaning it is at some position (i, l), it is going to move before any agent swaps, or delays, to position (i, l). Since the input stream is of bounded length $|R|$ and afterwards it is empty, there is at least one agent that can be moved. □

Now we look into simple streams and use lemmas 15 to 16 conclude that the queues in Algorithm 5 behave the same as the d values in Algorithm 4. Thus, it computes an optimum solution on simple streams.

Lemma 15. *In line 5.4, we have $q_l^{(i)} = \emptyset$ for all $i > 2|R|$.*

Proof. Every call to Algorithm 6 with a given row index i leaves in the lines 6.1, 6.4, 6.6, 6.10, 6.12, 6.13, 6.16, 6.19, 6.26, 6.28, 6.29, 6.42, 6.33, 6.36, 6.39, 6.40 and 6.41 at least one agent in row i if the input queues are not empty, or there are agents in R_i. Since $|R|$ rows contain at most $2|R|$ agents, no agent is delayed beyond row $2|R|$. □

Lemma 16. *If R is simple, then $t(a) = l$ for all $a \in q_l^{(i)}$ with $i \in \mathbb{N}$ and $l \in \{1, 2\}$.*

Proof. We need to consider all cases in Algorithm 8, besides line 8.8, because we assume that R is simple. In all these cases, Algorithm 8 pushes agents only the queues of their target lanes. There two more cases in which Algorithm 6 adds agents to queues. But, from R being simple follows, that in lines 6.10 and 6.26, the agent at r_1 has $t(r_1) = 1$. The same holds for the case $|q_1| > |q_2|$ in line 6.43, where $t(r_2) = 2$. Thus, the lemma statement holds until the row for which Algorithm 6 is called. For all further rows after the last call to to Algorithm 6, the lemma statement follows from the assumption (Algorithm 5) that the queues are initially empty and Algorithm 5 does not modify any further queue then the ones discussed Lemma 15. □

Lemma 17. *If R is simple then $d_{i,l} = |q_{i,l}|$.*

Proof. From Lemma 16 we know that every agent is assigned to its target queue in Algorithm 7. In Algorithm 6 we see (lines 6.1, 6.4, 6.6, 6.10, 6.12, 6.13, 6.16, 6.19, 6.26, 6.28, 6.29, 6.42, 6.33, 6.36, 6.39, 6.40 and 6.41) that the agents that are left on row i, i.e., (l_1, l_2) are the last elements of the queues, or the agents in row i in case there is no previous congestion. Meaning all, but one agent, for each lane is moved to the next row. The queues represent the congestion and the size of the queues increase, or decrease,

their size, like the d values increase, or decrease, in the first for loop of Algorithm 4. Note that the assignments in the lines 6.8, 6.22 and 6.25 of Algorithm 6 do not occur, since R is simple. □

Theorem 5. *Algorithm 5 computes an optimum solution for simple streams.*

Proof. Algorithm 5 only performs operations on agents when calling Algorithm 9. Delays are only performed if the agent is an element of a queue. Lemma 16 implies that this agent is removed from that queue after the operation in line 9.5 brings this agent to its correct lane. From Lemma 17 we know that the number of delays is the same as the sum of the d values of Algorithm 4. The optimality follows from Lemma 12. □

Lemma 18. *Let R be an agent stream and assume there exists exactly one i with $R_i = (2,1)$. If an optimum solution that does not delay any agent from $i-1$ to i exists, then also exists an optimum solution S with $S_i = (1,\perp)$ and $S_{i+1} = (\perp,2)$.*

Proof. Let R be an agent stream, such that there is exactly one i with $R_i = (2,1)$ and such that there is an optimum solution S', such that no agent coming from above into row i. Let $a_1 := R_{i,2}$ and $a_2 := R_{i,1}$. We note that $t(a_2, a_1) = (2,1)$. Let us assume there is an optimum solution, such that neither a_1, nor a_2 stay at row i. Without loss of generality, we assume a_2 is delayed first. But, then we can let a_1 stay at row i since there is no agent coming from above into row i. Swapping a_1 to lane 1 in row i and apply all other steps of the optimum solution creates a solution that uses at least one less delay, which is a contradiction to the optimality.

Thus, at least one agent of $\{a_1, a_2\}$ stays at row i. Without loss of generality, we assume that a_1 stays. Furthermore, assume a_2 delays at lane 1 until row j, then swaps at j and delays on lane 2 until j'. If we take the operations for the optimum solution, but we keep a_1 and a_2 in place. Then, the position $(j', 2)$ is empty and between row $i+1$ and j we have maximal $j-i-1$ agents, since only agents on lane 2 can stay there, all others need to move away when a_2 comes down.

So, we can use the delays we saved for not moving a_2 down, to delay the agents between row $i+1$ and j by using $j-i-1$ delays and row $i+1$ is now empty. So far we spend at least one delay less than the optimum solution. Thus, we can now either delay a_1, or a_2 to row $i+1$ and perform the swaps on rows i and $i+1$. □

Note that the proof of Lemma 19 considers Algorithm 4 and any optimum solution (and not necessarily the one by Algorithm 5). Theorem 7 uses this insight for arguing about the structural properties that an optimum can have in the presence of both 1-congested and 2-congested regions.

Lemma 19. *Let R be an agent stream and assume there exists exactly one index i with $t(R_i) = (2,1)$. Let R' be identical to R, besides $t(R'_i) = (1,2)$. Assume i is part of an 1-congested region, as well as a 2-congested region. Then, there exists an optimum solution S for R, which is also an optimum solution for R'.*

Proof. From Lemma 12 we know that Algorithm 4 computes an optimum solution S' for R'. In case there are agents in row $i+1$, Algorithm 4 is delaying them at least once to make space for the congestion above.

Let $(a_2, a_1) = R_i$. To construct an optimum solution for R, one uses the created free space in row $i+1$ to delay one agent from row i. This gives us the space to swap both agents, a_1 and a_2 and then move both to the to the same final position using the same steps as Algorithm 4 used. In fact this solution is identical to S'. □

Lemma 20. *Let R be an agent stream and assume that there exists exactly one i with $t(R_i) = (2, 1)$. Assume there exists an optimum solution that delays $k > 0$ agents from $i-1$ to i on one lane and that does not delay any agent on the other lane. Then there exists an optimum solution S with $S_{i,1} = R_{i,2}$ and $R_{i,1}$ delayed at least once and at most k times on lane 1 before it swaps.*

Proof. Let us assume we have an optimum solution S. Without loss of generality, we assume that there is at some point an agent at lane 2 and row $i-1$ that is delayed to row i. We do not loose generality, because if it would be on lane 1 it would be symmetric. However, throughout the solution S, there is no agent delayed on lane 1 from row $i-1$ to i. First, we assume that the agent $a_2 := R_{i,1}$ with target $t(a_2) = 2$ performs a swap at row i. This means that the agent $a_1 := R_{i,2}$ must delay on lane 2 at least to row $i+1$. Without loss of generality, we assume a_1 delays to row i'_1, then swaps there and delays to its final position $S_{i''_1,1}$ with $i''_1 \geq i'_1$. After its swap at row i, agent a_2 only delays until reaching its final position at $S_{i'_2,2}$ with $i'_2 > i$.

We can delay every agent between row $i+1$ and i'_1 by one for the same cost as delaying a_1 to row i'_1 and make space for the swap there. This makes space for delaying a_2 on lane 1 by one and then swap it on row $i+1$ with further delays afterwards until its final position. By the lemma assumption, no agent is delaying on lane 1 from row $i-1$ to row i and, thus, a_1 can stay at row i. But, this is a contradiction to the optimality of S, since this solution use at least on delay less. Therefore, a_2 cannot swap, as assumed, in row i and in an optimum solution a_1 stays in row i.

Delaying a_2 by one could recreate the same situation when the target of the agent on lane 2 is 1 as well, but the amount of agents delayed to row $i+1$ is one less, since there is no agent with target 2 on row i any more. This can be repeated at most k times. □

Definition 4. Let R be an agent stream. A row index j with $t(R_j) = (2, 1)$ is called *critical* (row index). We denote with $J(R)$ the set of critical row indices, or simply the set of critical rows. We define $q_{-1,1} = q_{-1,2} = \emptyset$. Let $j \in J(R)$. If $|q_{j-1,1}| > 0$ and $|q_{j-1,2}| > 0$, we say that j is a critical row index with *filled queues*. If $j \in J(R)$ and $|q_{j-1,1}| = 0$ and $|q_{j-1,2}| = 0$, then we say j is a critical row index with *empty queues*. Otherwise, we say j is a critical row index with *mixed queues*.

Lemma 21. *Let R be an agent stream. Let q be the queues computed by Algorithm 6 and d as computed by Algorithm 4. If all critical rows have empty queues, then $|q_{i,l}| \leq d_{i,l}$ for all i and for $l \in \{1, 2\}$.*

Proof. For simple inputs, the statement follows from Lemma 17. Assume the set of critical rows, $J(R)$, is not empty. Assume there are only critical rows with empty queues and let $j := \min J(R)$ be the first one. Then for all $j' < j$, we have $|q_{i,l}| = d_{i,l}$ as we saw in Lemma 17. Algorithm 4 ensures that $d_{j,1}$ and $d_{j,2}$ are larger or equal to 1 in line 4.7. Algorithm 6 does not delay both agents in R_j, but leaves one of these two agents at row j (line 6.1), namely the agent with target one, as we see in line 8.8 of Algorithm 8. Thus, $|q_{j,1}| < d_{j,1}$. For multiple occurrences, we repeat this argument and point out that for rows, that are not critical, the congestion based changes to the queues follow the changes of the d values, as we see in Lemma 17, i.e., $|q_{i-1,l}| - |q_{i,l}| = d_{i-1,l} - d_{i,l}$. □

Lemma 22. *Let R be an agent stream. Let q be the queues computed by Algorithm 6 and d as computed by Algorithm 4. Then $|q_{i,1}| + |q_{i,2}| \leq d_{i,1} + d_{i,2}$ for all i.*

Proof. For critical rows with filled queues, Algorithm 4 uses no additional delay, cf. line 4.7. It sorts the critical rows by using the space already created to solve the congestion above. Let us denote $R_j = (a_2, a_1)$. The direct solver would delay a_2 on lane 1, so a_1 can swap at row j (line 8.8 of Algorithm 8). Thus, $|q_{j,1}| = d_{j,1} + 1$ and $|q_{j,2}| = d_{j,2} - 1$. The crucial part is to see if the direct solver is able to leave at least as many agents in a row than Algorithm 4, as this is equivalent to looking at the size of the queues and, respectively, at the d values.

For each row in the solution S, as it is computed by Algorithm 5, there are at most two agents in this row, one on each lane. Let us consider a row index i, such that there is at least one agent in S_i. One of the agents in S_i is the last one of a queue, i.e., either $\mathit{last}(q_1^{(i-1)})$, or $\mathit{last}(q_2^{(i-1)})$. If there there are two agents in S_i, then we assume without loss of generality, that one of them was the last one in the queue of lane 1. If it had target 1, then the second agent is coming either from $q_1^{(i-1)}$, from $R_{i,1}$, from $\mathit{last}(q_2^{(i-1)})$, or there is none. If there is one agent place in row i, which is the last element of a queue, but it still needs to swap, then there is no second agent left in this row, as we see in lines 6.16 and 6.42 of Algorithm 6. The only case where an agent a with target l is added to the queue of lane $\bar{l}$, is the case in which $t(R_i) = (2, 1)$. This agent a continues to delay on the wrong lane if $t(R_{i,\bar{l}}) = l$ and the queues of lane $\bar{l}$ are still longer then on lane l, i.e., $|q_l^{(j)}| < |q_{\bar{l}}^{(j)}|$ for the next lanes $j > i$. Thus, for a given row, agents are delaying on the wrong side at most on one lane, but never on both, i.e., if for an index j there is an agent with target 2 in $q_1^{(j)}$ then there is none with target 1 in $q_2^{(j)}$. This means the direct solver can leave two agents in this row. Therefore, we can leave at least as many agents per row as the approximation algorithm does, which gives us the statement, $|q_{i,1}| + |q_{i,2}| \leq d_{i,1} + d_{i,2}$ for all i. □

Theorem 6. *Algorithm 5 computes the minimum makespan.*

Proof. Theorem 3 shows that Algorithm 4 computes the minimum makespan. From Lemma 22 we know that Algorithm 5 does not delay further than Algorithm 4 does. □

Lemma 23. *Let $l \in \{1,2\}$ and q be the queues after executing Algorithm 5 until line 5.3. If there exists an agent $a \in q_l^{(i)}$ with $t(a) = \bar{l}$, then there is no agent $a' \in q_{\bar{l}}^{(i)}$ with $t(a') = l$.*

Proof. We discuss two cases, first both agents where in the same row initially, e.g., $R_i = (a, a')$, then the case that they did not share the same row in R.

Neither Algorithm 8, nor Algorithm 6 move an agent a into $q_l^{(i)}$ with $t(a) = \bar{l}$ and agent a' into $q_{\bar{l}}^{(i)}$ with $t(a') = l$ in the same iteration, i.e., a and a' are added during different calls to Algorithm 6 using different row indices. More precisely, since we do not swap an agent that is already on its target lane to the opposite lane, the only case we need to look into is $t(R_i) = (2,1)$. But, this is in general handled in line 8.8, where indeed the agent on lane 2 with target 1 is added to the queue of lane 1. For the special cases of lines 6.40, 6.26 and 6.10 (and the occurrences in the symmetric part, e.g., line 6.43), one agent is swapped to its target lane. Thus, Algorithm 6 has not added them in the same row and they must be added in separate rows, i.e., one agent was in the queue on the not-target lane and one in the row when calling Algorithm 6.

To conclude the lemma, we need to look into the case the agents a and a' are added during different calls to Algorithm 6. We assume there is an agent $a \in q_l^{(i-1)}$ with $t(a) = \bar{l}$ and no such agent in $q_{\bar{l}}^{(i-1)}$. If there is no such row, then the lemma statement follows immediately. To add agent a' into $q_{\bar{l}}^{(i)}$ with $t(a') = l$ it needs to be in $R_{i,\bar{l}}$. There are two possible reasons for a' to be added to $q_{\bar{l}}^{(i)}$. Either, $R_{i,l}$ stays in place, or $R_{i,l}$ swaps to lane $\bar{l}$. The first case cannot happen because $|q_l^{(i-1)}| > 0$ and the second case implies that all agents in $q_l^{(i-1)}$ with target lane $\bar{l}$ because a' already paid for moving away, but then there is no agent $b \in q_l^{(i)}$ with $t(b) = \bar{l}$. □

Theorem 7. *Algorithm 5 computes an optimum solution.*

Proof. The cost of an optimum solution is lower bounded by the cost of congestion, i.e., the cost of solving an input where all critical rows are swapped in place before solving it, i.e., replace $R_i = (a, b)$ such that $t(a, b) = (2, 1)$ with (b, a). The additional costs, compared, to the congestion, come from solving critical rows. In the situation that both agents in a critical row need to delay anyway due to congestion, the space for swapping is generated for free.

For simple inputs, we showed optimality in Theorem 5. Assume we only have critical rows with empty queues. Then, the solution of the critical rows does not affect each other and the direct solver is constructing the solution of Lemma 18. If we have additionally rows with full queues, we see that the direct solver constructs a solution as in Lemma 19. In fact, this case can be solved without using more delays than those required to solve congestion as we saw in the proof of Lemma 22.

The interesting case is when the we have critical rows with mixed queues. Here, we want to apply Lemma 20. Let $(a_2, a_1) := R_i$ be this row and let, without loss of generality, $|q_2^{(i-1)}| > 0$. We know from Lemma 20 that a_2 is delaying at least once, which also means an agent at $R_{i+1,1}$ needs to move away. If the next row is empty, a_2 just swaps to lane 2

and no unnecessary delays have been executed, since there were agents delaying on lane 2. Similarly, if there is an agent on $R_{i+1,2}$ with target lane 2. This agent delays to make space for a_2. If there is an agent $a_3 := R_{i+1,2}$ with $t(a_3) = 1$, then there are three cases. The first is that a_3 swaps and delays before a_2 comes down, the second that a_3 delays on lane 2 and a_2 swaps on lane $i + 1$ and the third that a_2 delays further on lane 1 and a_3 swaps afterwards.

If $|q_1^{(i+1)}| > 1$ then all other agents besides a_2 in $q_1^{(i+1)}$ have target lane 1, which is implied by Lemma 23. If we assume $|q_1^{(i+1)}| > 2$, then it follows that $|q_2^{(i)}| > 1$ and at least one agent in $q_2^{(i)}$ has target lane 1. The only case in which such an agent delays on lane 2 is that $t(R_{i-1,1}) = 2$ and $q_1^{(i-1)}$ is either empty, or contains one agent with target lane 1. Since $R_{i-1,1}$ swapped $|q_1^{(i-1)}| = |q_2^{(i-1)}|$, but then we can have at most two agents with target lane 1 in $q_2^{(i-1)}$. Since every agent with target lane 1 swapped in row i there is no such agent in $q_2^{(i)}$. We also see, if we denote with $j > i$ the row index in which a_2 swaps to lane 2, no additional agents gets added to $q_2^{(j)}$. Because, $t(R_{j,2}) = 1$ and it swaps in row j, otherwise a_2 would swap into the free space in row j. This goes on until $|q_1^{(j)}| = |q_2^{(j)}|$, where a_2 swaps lane.

We show optimality by showing that we can apply Lemma 20 and that we do not do unnecessary operations within the bounds of Lemma 20. The problem occurs if we have agents with some target l in some queue $q_\perp^{(i)}$, but we need to leave the output at $(i-1, l)$ empty, i.e., $S_{i-1,l} = \perp$, meaning this agents could have stayed. This can happen for a critical row, where one of the agents must delay on the wrong side, as well as afterwards, where such an agent stays on the wrong side for some more rows. For the critical row, we have two choices of who is delaying, but from Lemma 20, we know who it is. For the case an agent was in a queue and stayed in the wrong lane, this can only happen in lines 6.10 and 6.26 of Algorithm 6. This is the case if we have a row j such that some $a_2 \in q_1^{(j)}$ exists with $t(a_2) = 2$ and $t(R_{j,2}) = 1$ and $|q_1^{(j)}| \leq |q_2^{(j)}|$. If $t(R_{j,2}) \neq 1$, then we can sort all queues and all delays are based on congestion and therefore we use the minimum amount, besides if there is now agent with target 1 after a_2 in $q_1^{(j)}$, but at least one before. This means they have to delay one more to let a_2 through and, one could say, this artificially increases the congestion on lane 1 by one. Nevertheless, a_2 has overtaken a agent with target lane 1 on every lane it delayed. If a_2 would swap earlier to lane 2, then this would require one of these agents on lane 2 to delay to provide space, but this would have created real congestion and therefore would have created at least the same cost.

From Lemma 23 we know that there is no agent in $q_2^{(j)}$ with target lane 1. Thus, as long as $|q_1^{(j)}| \leq |q_2^{(j)}|$, we can leave the last element of $q_2^{(j)}$, as well as $R_{j,2}$ in row j. If now $|q_1^{(j)}| > |q_2^{(j)}|$ and if there is an agent a_4 with target lane 1 after a_2 in $q_1^{(j)}$, then two agents can be left in row j without extra cost, i.e., every agent in $q_2^{(j)}$ delays further to row $j + 1$, then a_2 swaps, and $S_j = (a_4, a_2$. If not and if there are agents with target 1 in $R_{j,2}$, or before a_2 in $q_1^{(j)}$, then they stay in row j, since if a_2 stays, it is not possible to let another agent stay. If this is not possible then a_2 stays in row j. Then additional

delays to make space for a_2 might be required to make space to swap lanes. This might create, as in the previously discussed case, congestion in the following lanes.

Thus, all alternatives ways in resolving a case with $t(R_i) = (2, 1)$ results in at least the same number of congestion and additional delays for navigation in the successive rows and, therefore, we can conclude the optimality of Algorithm 5. □

5.4 Optimum Agent Sorting is NP-hard: the Proof Details

In this section, we will study what happens when we generalise the OAS problem from two lanes to multiple lanes. Assume that we have ℓ lanes, and the agents are labelled with targets $\mathcal{S}' := \{1, 2, \ldots, \ell\}$ by the mapping $t : A \mapsto \mathcal{S}'$. With A we denote again the set of agents. The stream is then given by $R \in (A \cup \{\perp\})^{\mathbb{N}\times\ell}$, where is agent $a \in A$ occurs only once in R. With cell we denote the positions $R_{i,l}$.

The stream operations are a natural generalisation of the two-lane case: we can *delay* an agent, if there is empty space behind it, and we can move an agent sideways if there is empty space in an adjacent lane. We can only move agents sideways towards their target lane, not away from it. We will now prove that minimising the total cost for this generalisation is NP-hard.

A vertex cover of graph $G := (V, E)$ is a set of vertices $V' \subseteq V$, such that each edge of G is incident to at least one vertex in V'. The minimum vertex cover problem deals with finding the smallest vertex cover, V', in graph G and it is NP-hard [10]. Theorem 8 shows that the studied problem is reducible to the minimum vertex cover problem. For the sake of simple presentation of the proof, throughout this section, we assume that graph G is 3-regular [6] and that it has an even number, $N = |V|$, of vertices.

Theorem 8 (A Reduction from the Minimum Vertex Covering problem)**.** *For any even integer number $N \in \mathbb{N}$ and a 3-regular graph $G = (V, E)$ that has N vertices: (1) there is a polynomial time construction of the stream R (Section 5.4.1), according to $G = (V, E)$ (2) if there is a polynomial time and exact algorithm, A_o, that solves R, we can present an algorithm that takes A_o's output and provides an optimum solution for the constructed stream R within a polynomial time (Algorithm 10), and (3) there is a polynomial time transformation from A_c's solution for R, to a solution of the Minimum Vertex Covering problem for $G = (V, E)$.*

5.4.1 The construction of the stream R, for a given graph $G := (V, E)$

For any given even number $N \in \mathbb{N}$ and a 3-regular graph $G = (V, E)$ with $V = \{1, \ldots, N\}$ as the set of vertices, we construct the stream R, with $\ell = 18N/2 + 1$ lanes and $\mathcal{O}(N^3)$ rows in which we place agents, see Figure 5.5. The stream R is the basis of our reduction proof (Theorem 8). We start by describing the stream details and their polynomial time construction (Lemma 24) before presenting A_c and showing its optimality.

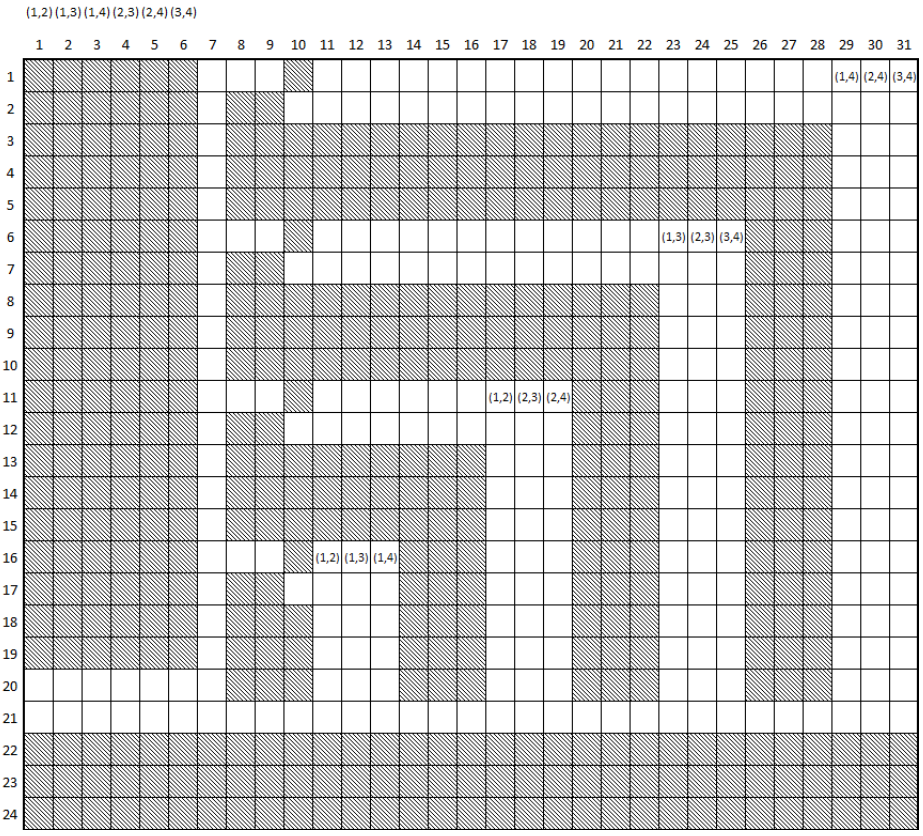


Figure 5.5: Stream constructed for the case of a 3-regular graph with $N = 4$ vertices, cf. graph G, Figure 5.6

5.4.2 Moving agents to their target lanes at the lower and upper cavities

In Figure 5.5, the painted cells refer to agents that are placed in their target lanes, i.e., $R(x)$'s solution places them in their current lanes. The agents that are not placed in their lanes are marked by their lane tags that refer to edges. This associates vertex $k \in V$ and its edges, with a set of three agents $E(k) = \{(k, \bullet) \in E\}$. Note that the agents' target lanes are at the first (from the left) $3N/2$ lanes, cf. the top left part of Figure 5.5.

We assume that the $5(N-k)$-th (from the top) row includes the agent $(k, k_i) \in E(k) : k \neq k_i \wedge i \in \{1, 2, 3\}$ at the cells on the lanes $x + i$ according to their lexical order, where $x = 3N/2 + 6k - 2$. Note that each agent (k, k_i) that is not placed on its target lane, has an open path to the *lower cavity*, which includes the $3N/2$ first (most left) at the $5N + 1$ row (from the top). Moreover, (k, k_i) can also move to the *upper cavity*, which is one row above the lower cavity, i.e., row $5N$. To that end, the algorithm can delay a special agent, which we call the N-th *valve*, on the $3N/2 + 4$ lane (and perhaps there is also a need to move agent $(k, k_j) : j < i$, if exists). By that, the algorithm opens a path to the upper cavity, see for example the 4-th valve on the first row (lane $3N/2 + 4$), which has no agent behind it on the 2-nd row.

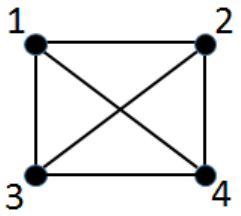


Figure 5.6: Graph G, which is 3-regular and has $N = 4$ vertices.

5.4.3 Motivating the wall thickness

Observe the set of agents that are placed between the paths that lead the agents $(k, \bullet) \in E(k)$ to the lower and upper cavities, which appear as painted cells in Figure 5.5. We use the name the *(inner) wall (of vertex k)* when referring to this set. The proof shows (Lemma 26) the existence of an exact algorithm that moves all agents to their target lanes at the lower and upper cavities. Moreover, it shows (Theorem 9) that this exact algorithm never moves agents that are already in their target lanes. For the sake of simple proof presentation, starting from the $4N + 2$ row, our construction includes $\mathcal{O}(N^3)$ rows of agents that are placed in their target lanes, cf. the rows that appear immediately below the lower cavity. Moreover, the (inner) walls between the paths (from above and from below) to the cavities are at least three cell thick.

5.4.4 Estimating the running time of the construction procedure

Lemma 24 shows that the stream R can be constructed within polynomial time.

Lemma 24 (Polynomial construction time of **R**)**.** *For any given $N \in \mathbb{N} : N/2 \in \mathbb{N}$ and a 3-regular graph $G = (V, E)$, we can construct stream R in $\mathcal{O}(N^4)$ time.*

Proof. We start by showing how to construct the stream R, before estimating the running time of this construction procedure. Observe that Figure 5.5 depicts a somewhat recursive structure. Namely, when considering the vertices in the graph $G := (V, E)$, we can extend the construction in Figure 5.5, which considers the case of $N = 4$. Suppose we have constructed a stream for the case of $N = 4$ and now we wish to extend to the case of $N = 5$. Let us do that by simply adding five more rows on the top and six more lanes on the right as well as three move lanes between lane $3N/2$ and $3N/2 + 1$. Moreover, we perform this extension by taking the following steps (i) to (iii). The construction for the cases of $N = 6, 7, \ldots$ is done in a similar manner after reaching the $(N - 1)$-th stage.

i We lay the N-th (inner) wall (which has the N-th valve) that is dedicated to the added vertex. We do that while keeping the lower and upper paths to the newly added target lanes in the lower and upper cavities.

ii We place agents in painted cells (and by that mark that they have reached their target lanes) at the rows that start at the $4N + 2$ row (and then below) as well as in the upper left corner of the stream (where we have just added more rows).

iii We make sure that the labels of agents $(k, \bullet) \in E(k)$ and the first $(3N/2)$ (left most) lanes are according to the edges in the graph that considers $N = 5$ nodes.

In order to estimate the running time of the above procedure, we say the following: (1) copying the array that stores the stream R, to an array that stores $R(N + 1)$ can be done in $\mathcal{O}(N^3)$ time, because of the array dimensions are in $\mathcal{O}(N^3)$, (2) running the above procedure can be done in polynomial time, because steps (i) to (iii) require each $\mathcal{O}(N^3)$ time, and (3) starting from $N = 2$, we can construct $R : N/2 \in \mathbb{N}$ using a liner number of times in which we run steps (i) to (iii). □

5.4.5 Optimum placement functions, ϕ, and paths that lead to them

Suppose that there is an algorithm A_o that solves the Optimum Agent Sorting problem optimally and runs in polynomial time. We show that A_o's solution to stream R, places agent (k, k_i) either in the lower or the upper cavities, where $k \in V$, $i \in \{1, 2, 3\}$ and $(k, k_i) \in E(k_i)$ is the i-the element in $E(k)$ when considering an acceding lexical order. We define the *optimum placement* function $\phi(k, i) \in \{0, 1\}$ as one that returns 1, if and only if, the solution of algorithm A_o places (k, k_i) on a higher row than $(k_i, k) \in E(k_i)$. We denote A_o's solution by the $N \times 3$ matrix $\Phi = [\phi(k, i)]$. Note that the definitions of ϕ and Φ do not consider the existence of the lower and upper cavities. Lemma 25 shows that we can construct Φ within a polynomial time.

Lemma 25. *Once A_o solves R, a polynomial time algorithm can construct Φ.*

Proof. Let $\xi(k, k')$ be the row number that agent $(k, k') \in E(k)$ ends up at when A_o solves R. There are $\mathcal{O}(N^2)$ cells to scan and to decide accordingly about the value of $\phi(k, k')$, i.e., $\phi(k, k') = 1$ if $\xi(k, k') < \xi(k', k)$ and $\phi(k, k') = 0$ otherwise. □

We say that a sequence of moves of a particular agent is *simple* when every move is only space dependent, rather than agent dependent. Namely, it depends only on the fact that there are empty cells along the path that the agent moves to rather than the moves that other agents take in order to create the needed space.

We consider two examples of simple paths:

1. The *lower path* is a simple path that leads agent $(k, \bullet)$ to its target placement at the lower cavity. In detail, let S_ℓ be the (simple) sequence of moves in which $(k, \bullet)$ first takes $5k$ delays and then $(3N/2 + 1 - plc(k, k')) + (6k - 2 + ind(k, k'))$ swaps to the left until it reaches its placement according to Φ in the lower cavity, where $ind(k, k')$ is the order of $(k, \bullet)$ in $E(k)$ and $plc(k, k')$ is $(k, \bullet)$'s order in Q.

2. After delaying the $k - th$ valve once, the same agent has access to another simple path, which is the *upper path* that leads it to the upper cavity. In detail, let S_ℓ be the (simple) sequence of moves in which $(k, \bullet)$ first takes $(6k - 2 + ind(k, k'))$ swaps to the left, $5k - 1$ delays and then $(3N/2 + 1 - plc(k, k'))$ swaps to the left until it reaches its placement according to Φ in the upper cavity.

Algorithm 10: A_c is an algorithm that the proof shows to comply with A_o's solution Φ as well as to be optimum.

10.1 **constants:**
10.2 Q is the (lexicographic and ascending) ordered sequence of E;
10.3 $\forall k \in V$, $E(k)$ is set of the agents (labelled by the edges of vertex $k \in V$);
10.4 **functions:**
10.5 $ind(k,k')$ returns i when (k,k') is the i-th agent in $E(k)$;
10.6 $plc(k,k')$ returns i when (k,k') is the i-th edge in Q;
10.7 **input:**
10.8 The optimum placement function, $\phi()$, which defines the matrix Φ;
10.9 **for** $k := 1$ **to** N **do**
10.10 **if** $\exists (k,k') \in E(k) : \phi(k,k') = 1$ **then delay** valve k **once**;
10.11 **foreach** (k,k') *edge in* Q *(by the lexical ascending order of* E **do**
10.12 **let** (k,k') and (k',k) two agents, such that $(k,k') \in E(k) \wedge \phi(k,k') = 0 \wedge$ $(k',k) \in E(k') \wedge \phi(k',k) = 1$; `/* (k,k') is to be placed at the lower cavity and (k',k) in the upper one */`
`/* move (k,k') along the lower path. Note that` $\phi(k,k') = 0$. `*/`
10.13 **delay** (k,k') **for** $5k - \phi(k,k')$ **times**;
10.14 **swap left** (k,k') **for** $(3N/2 + 1 - plc(k,k')) + (6k - 2 + ind(k,k'))$ **times**;
`/* move (k',k) along the upper path. Note that` $\phi(k,k') = 1$. `*/`
10.15 **swap left** (k,k') **for** $(6k - 2 + ind(k,k'))$ **times**;
10.16 **delay** (k',k) **for** $5k - \phi(k,k')$ **times**;
10.17 **swap left** (k,k') **for** $(3N/2 + 1 - plc(k,k'))$ **times**;

5.4.6 A_c complies with A_o's solution and it is optimum

A_c takes the function Φ as an input and leads the agents to the lower and upper cavities according to Φ. In detail, A_c delays exactly once any k-th valve for which there is at least one agent (k,k'), such that $\phi(k,k') = 1$ (line 10.10). It then scans all the edges of graph G by their ascending lexical order (line 10.11). Note that, by the construction of the flow R, there are two agents associated with each edge (line 10.12). One of these agents is to move along the lower path to the lower cavity (lines 10.13 to10.14) and another to the upper cavity via the upper path (lines 10.15 to10.17).

Lemma 26. *A_c complies with A_o's solution Φ.*

Proof. We show that A_c moves agent $(k,\bullet)$ to the lower cavity when $\phi(k,\bullet) = 0$ and to the upper one when $\phi(k,\bullet) = 1$. We do that by demonstrating that A_c allows a single agent at a time to reach its placements while using $\phi()$'s value for choosing the transit path.

We note that, according to $\phi(k,\bullet)$, A_c: (1) makes sure that the k-th valve is not blocking the upper path for $(k,\bullet)$ to the upper cavity (line 10.10), and (2) chooses whether to take the lower or the upper path (line 10.12 to 10.17), which can lead agent $(k,\bullet) \in E(k)$ to the lower, and respectively, to the upper cavity. Whenever A_c moves agent $(k,\bullet)$ using these paths, only that agent is moving (line 10.13 to 10.17). Moreover, A_c selects

these agents according to the ascending lexical order of the edges in E (line 10.11), which is exactly the same order in which the first $3N/2$ (from the left) lanes appear in the stream R, (Figure 5.5). Note that each such lane has one available cell in the lower cavity and one in the upper. Moreover, there are exactly two agents for each edge (because each edge has two vertices). Thus, during the movement of agent $(k, \bullet)$, indeed these paths stay simple (with respect to agent $(k, \bullet)$), because we never have the case in which there is an agent that blocks the way of agent $(k, \bullet)$ or that there is no space left for agent $(k, \bullet)$ in its cavity. To end this proof, we remind that A_c places each of these two agents in a way the complies with Φ. □

Theorem 9 ($\mathbf{A_c}$ is optimum)**.** *The numbers of swap and delay moves that A_c takes for solving R are not larger than the numbers of moves that A_o takes to solve R (while complying with the same solution, Φ).*

Proof. We start the proof by noting that the only moves that A_o takes are of delay and swap to the left. Thus, the number of swap to the left moves that A_o and A_c take is the same, because both algorithms need to comply with the same solution, Φ. Thus, the rest of the proof focuses merely on showing that A_o cannot take fewer delay moves than A_c.

Remark 1. By the problem definition, once A_o delays agent $(k, \bullet)$ at row x, it holds that agent $(k, \bullet)$ never return to row x.

Let $\xi(k, k')$ be the row number that agent $(k, k') \in E(k)$ ends up at when A_o solves R. Lemma 27 shows that A_o moves (k, k') either to the lower or the upper cavities.

Lemma 27. $\xi(k, k') \in \{5N, 5N + 1\}$

Proof. Suppose this lemma is false, i.e., A_o takes fewer delay moves than A_c and yet $\exists(k, k') \in E(k)$, such that: (1) $\xi(k, k') > 5N + 1$ or (2) $\xi(k, k') < 5N$. We show that both case (1) and (2) lead the proof of this lemma to a contradiction.

The case of (1): $\xi(\mathbf{k}, \mathbf{k}') > \mathbf{5N + 1}$. Recall that R has $\mathcal{O}(N^3)$ rows that start immediately after row $5N + 1$ (Section 5.4.1). Therefore, in order to make a single cell available for agent (k, k') at row $\xi(k, k') > 5N + 1$, algorithm A_o takes at least $\mathcal{O}(N^3)$ delay moves. We show that this leads case (1) to a contradiction by arguing that A_c takes fewer than $\mathcal{O}(N^3)$ delay moves for agent (k, k') or any other agent. This is true because agent (k, k') takes at most $(5N+1)$ delay moves and there are $3N$ agents.

The case of (2): $\xi(\mathbf{k}, \mathbf{k}') < \mathbf{5N}$. Let K be the set of agents (k, k') for which $\xi(k, k') < 5N$, and $pre(K) \geq \sum_{(k,k')\in K}(5N - \xi(k, k'))$ (preparation) be the sum of the number of delay moves that A_o takes in order to prepare the cell-space needed above row $5N$ for all the agents in K. By Remark 1, the number of delays that A_o takes for all agents $(k, k') \in K$ is $delay_{A_o}(K) = \sum_{(k,k')\in K}(\xi(k, k') - (5(N - k) + 1))$. Let $delay_{A_c}(K) = \sum_{(k,k')\in K}(5k - \phi(k, k'))$ be the number of delay steps that A_c takes for all agents $(k, k') \in K$, cf. lines 10.13 and 10.16.

On the one hand, in order to make the cells above row $5N$ available for all the agents in K, A_o prepare space for the agents in K by delaying agents (that are not necessarily in K)

$pre(K)$ times plus bringing the agents in K to their rows using $delay_{A_o}(K)$ delays. On the other hand, A_c delays the agents $(k,k') \in K$ at most $delay_{A_c}(K)$ times. Therefore, by the assumption that appears at the start of this proof that the lemma is false, the former sum is strictly less than the latter one, cf. Equation (5.4).

$$
\begin{aligned}
pre(K) + delay_{A_o}(K) &< delay_{A_c}(K) \qquad (5.4)\\
\sum_{(k,k')\in K} (5N - \xi(k,k')) + (\xi(k,k') - (5(N-k)+1)) &< \sum_{(k,k')\in K} (5k - \phi(k,k'))\\
(5N - \xi(k,k')) + (\xi(k,k') - (5(N-k)+1)) &< 5k - \phi(k,k')\\
5N - \xi(k,k') + \xi(k,k') - 5(N-k) - 1 &< 5k - \phi(k,k')\\
5N - \xi(k,k') + \xi(k,k') - 5N + 5k - 1 &< 5k - \phi(k,k')\\
5N - 5N + \xi(k,k') - \xi(k,k') + 5k - 1 &< 5k - \phi(k,k')\\
-1 &< -\phi(k,k')\\
\phi(k,k') &< 1
\end{aligned}
$$

Recall that $\phi(k,k') \in \{0,1\}$ returns 1, if and only if, the solution of algorithm A_o places (k,k') on a higher row than (k',k). Equation 5.4 implies that $\phi(k,k') = 0$ and that $(k',k) \in K$ (in addition to the assumption that $(k,k') \in K$), because both (k,k') and (k',k) have the same target lanes, which in turn implies that $\phi(k',k) = 0$. The latter contradicts $\phi()$'s definition and the fact that only one agent can be placed in any cell. □

We say that path p_1 incompletely follows path p_2 when an agent that follows p_1 indeed passes through some of the cells in p_2 (other than perhaps the source and destination cells) but does not exactly follow p_2 from source to destination. Let K_{lower} be the set of agents $(k,k') \in E(k)$ for which (1) $\phi(k,k') = 0$ as well as (2) when A_o move agent (k,k') to its final placement, it incompletely follows the lower path (nor does it move (k,k') exactly along the upper path after delaying the k-th valve once) and yet A_o delays agent (k,k') fewer times than A_c. Let $K_{upper+valve}$ be the set of agents $(k,k') \in E(k)$ for which (1) $\phi(k,k') = 1$, as well as (2) when A_o moves agent (k,k') to its final placement it incompletely follows the upper path (with or without delaying the k-th valve) nor does it move (k,k') exactly along the lower path (if that is at all possible) and yet A_o delays agent (k,k') fewer times than A_c. Lemma 28 shows that these two sets, K_{lower} and $K_{upper+valve}$, must be empty and thus the deviations from the lower, and respectively, the upper paths are not possible for these cases.

Lemma 28. *(1)* $K_{lower} = \emptyset$ *and (2)* $K_{upper+valve} = \emptyset$.

Proof. Note that $\phi(k,k') = 0$ and $\phi(k,k') = 1$ imply that agent (k,k') ends up at the lower, and respectively, the upper cavities (Lemma 27 and the definition of the $\phi()$ function). Moreover, the lower and the upper paths have an optimum number of delay moves to the lower, and respectively, upper cavities (when assuming that no valve blocks the way). This is true because the number of delay (and swap) moves matches

the (Manhattan) distance from source to destination. Call the latter fact, the *optimum distance argument*.

Part (1). Note that agent (k, k') can reach the lower cavity via the upper path only after delaying once the k-th valve. However, this cannot be done with fewer delay moves than the ones in the lower path (due to the Manhattan distance from source to destination). Thus, this part of the proof is true by the optimum distance argument.

Part (2). By Remark 1, agent (k, k') cannot reach the upper cavity via the lower path. Thus, this part of the proof is true by the optimum distance argument. □

We say that paths p_1 and p_2 are disjoint when an agent that follows p_1 does not passes through any of the cells in p_2 (other than perhaps the source and destination cells). Let $K_{upper-valve}$ be the set of agents $(k, k') \in E(k)$ for which (1) $\phi(k, k') = 1$, (2) A_o does not delay the k-th valve before delaying (k, k'), as well as (3) A_o moves (k, k') to its final placement over any path that is disjoint with the upper path and yet A_o delays agent (k, k') fewer times than A_c. Lemma 29 shows that $K_{upper-valve}$ must be empty.

Lemma 29. $K_{upper-valve} = \emptyset$.

Proof. Suppose that this lemma is false. Namely, $K_{upper-valve} \neq \emptyset$ and yet A_o delays (k_{first}, k'_{first}) fewer times than A_c, where $(k_{first}, k'_{first}) \in K_{upper-valve}$ is the first agent when ordering $K_{upper-valve}$ according to Q (line 10.2).

• **A_o must move (k_{first}, k'_{first}) below the row of the k-th valve.**

Note that, in R's initial state, the k_{first}-th valve and agent (k_{first}, k'_{first}) share the same row. By item (2) in $K_{upper-valve}$'s definition, A_o does not delay the k_{first}-th valve before delaying (k_{first}, k'_{first}). Therefore, A_o delays agent (k_{first}, k'_{first}) and by that, it leaves the row of the k-th valve. Moreover, agent (k_{first}, k'_{first}) never return to that row (Remark 1).

• **A_o must move (k_{first}, k'_{first}) above the row of the lower cavity.**

By item (1) in the definition of $K_{upper-valve}$, $\phi(k_{first}, k'_{first}) = 1$ and thus A_o moves (k_{first}, k'_{first}) to the upper cavity (Lemma 28). Once A_o moves (k_{first}, k'_{first}) to the row of the lower cavity this proof is done, because it cannot move it to the upper cavity (Remark 1).

• **Demonstrating a contradiction.**

Let us consider the path along which A_o moves agent (k_{first}, k'_{first}), i.e., starting from its initial position to lane $3N/2 + 1$. Notice the walls (of vertices $k'' \in \{1, \ldots, k_{first}\}$) along with any such path. By lemmas 27 and 28 as well as the two items above, agent (k_{first}, k'_{first}) reaches lane $3N/2 + 1$ via a path that stretches between the rows of the k_{first}-th valve and of the lower cavity in a way that must go through these k''-th walls. Note that breaking through each of these k''-th walls requires A_o to delay at least three agents that have already reached their target lanes.

By similar arguments to the ones that appear above regarding (k_{first}, k'_{first}), the total number of times that A_o delays agents that have already reached their target lanes is at least $(3 \cdot |K_{upper-valve}|/3)$ (because at least one agent in $E(k_{first})$ has to break through wall of k_{first} so that at most three agents could travel through this opened path at the cost of three delays). In addition to these delays, A_o has to move all the agents in

$K_{upper-valve}$ and perform a number of delays that is not smaller than the number that A_c takes when moving these agents (due to the fact that both algorithms move them over the same Manhattan distance). Thus, we have reached a contradiction, because A_o takes at least $|K_{upper-valve}|$ more delays than A_c and $|K_{upper-valve}| \geq 1$ (by the statement of this lemma). □

□

Lemma 30. *A_c has a polynomial running time.*

Proof. Line 10.10 runs in $\mathcal{O}(N)$ time. Line 10.11 considers $\mathcal{O}(N^2)$ agents and A_c moves each agent over at most $\mathcal{O}(N^2)$ cells. A_c's running time is in $\mathcal{O}(N^4)$. □

5.4.7 The reduction proof

The proof of Theorem 8. We demonstrate a polynomial time construction of stream, R, in Lemma 24, i.e., part (1) is correct. Part (2) is correct, because Lemma 26 shows that A_c complies with the solution of A_o and Lemma 26 shows that this solution is optimum.

Let V' be the set of vertices $k \in V$, such that A_c delays the k-th valve when solving R. We show that Part (3) is correct, by showing that V' is a minimum vertex cover solution for $G := (V, E)$ and by showing the total running is polynomial.

• **The set V' is a minimum vertex cover solution for $G := (V, E)$.** By the construction of R, (Section 5.4.1) and the fact that A_c solves the studied problem (Lemma 26), the upper cavity includes a set of target lanes (of agents) that includes all the edges in E of graph $G := (V, E)$. Namely, V' is a cover of the edges E in G. Recall that A_c is optimum (Part (2) of this proof), and that it delays the k-th valve only when there is at least one agent $(k, k') \in E(k)$ that it moves to the upper cavity (line 10.10). Note that any other agent $(k, k'') \in E(k)$ can move to the upper cavity at a lower cost than to the lower one (as long there is space available in the upper cavity, i.e., agent (k'', k) is not already there). This means that $k \in V'$ if, and only if, agent (k, k') or (k', k) ends up at the upper cavity and the latter occurs if, and only if, the optimum solution, which A_c provides, decides to delay the k-th valve.

• **The total running time is polynomial.** We can solve R, using A_o in polynomial time (by assumption), construct Φ in polynomial time (Lemma 25), run A_c in polynomial time (Lemma 30) and then construct V' in polynomial time (because there are N valve cells to check and their locations in R is known). □

Bibliography

[1] Maksat Atagoziyev, Klaus W. Schmidt, and Ece G. Schmidt. "Lane Change Scheduling for Autonomous Vehicles". In: *IFAC-PapersOnLine* 49.3 (2016). 14th IFAC Symposium on Control in Transportation SystemsCTS 2016, pp. 61 –66. ISSN: 2405-8963. DOI: http://dx.doi.org/10.1016/j.ifacol.2016.07.011. URL: http://www.sciencedirect.com/science/article/pii/S2405896316302063.

[2] Wonshik Chee and Masayoshi Tomizuka. "Vehicle lane change maneuver in automated highway systems". In: *California Partners for Advanced Transit and Highways (PATH)* (1994).

[3] Y. Fang, F. Chu, S. Mammar, and M. Zhou. "Optimal Lane Reservation in Transportation Network". In: *IEEE Transactions on Intelligent Transportation Systems* 13.2 (June 2012), pp. 482–491. ISSN: 1524-9050. DOI: 10.1109/TITS.2011.2171337.

[4] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. "On graph problems in a semi-streaming model". In: *Theor. Comput. Sci.* 348.2-3 (2005), pp. 207–216. DOI: 10.1016/j.tcs.2005.09.013. URL: https://doi.org/10.1016/j.tcs.2005.09.013.

[5] Li Feng, Li Gao, and Yun-Hui Li. "Research on Information Processing of Intelligent Lane-Changing Behaviors for Unmanned Ground Vehicles". In: *Computational Intelligence and Design (ISCID), 2016 9th International Symposium on*. Vol. 2. IEEE. 2016, pp. 38–41.

[6] Gerd Fricke, Stephen T. Hedetniemi, and David Pokrass Jacobs. "Independence and Irredundance in k-Regular Graphs". In: *Ars Comb.* 49 (1998).

[7] C. Hatipoglu, U. Ozguner, and K. A. Unyelioglu. "On optimal design of a lane change controller". In: *Intelligent Vehicles '95 Symposium., Proceedings of the*. Sept. 1995, pp. 436–441.

[8] Ding-wei Huang. "Lane-changing behavior on highways". In: *Physical Review E* 66.2 (2002), p. 026124.

[9] Wm. Woolsey Johnson and William E. Story. "Notes on the "15" Puzzle". English. In: *American Journal of Mathematics* 2.4 (1879), pp. 397–404. ISSN: 00029327. URL: http://www.jstor.org/stable/2369492.

[10] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York.* Ed. by Raymond E. Miller and James W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103. ISBN: 0-306-30707-3. URL: http://www.cs.berkeley.edu/~luca/cs172/karp.pdf.

[11] Jorge A. Laval and Carlos F. Daganzo. "Lane-changing in traffic streams". In: *Transportation Research Part B: Methodological* 40.3 (2006), pp. 251 –264. ISSN: 0191-2615. DOI: http://dx.doi.org/10.1016/j.trb.2005.04.003. URL: http://www.sciencedirect.com/science/article/pii/S019126150500055X.

[12] Kai Nagel. "High-speed microsimulations of traffic flow". PhD thesis. University of Cologne, Germany, 1995. URL: http://d-nb.info/943852641.

[13] Kai Nagel, Dietrich E Wolf, Peter Wagner, and Patrice Simon. "Two-lane traffic rules for cellular automata: A systematic approach". In: *Physical Review E* 58.2 (1998), p. 1425.

[14] José Eugenio Naranjo, Carlos González, Ricardo García, and Teresa de Pedro. "Lane-Change Fuzzy Control in Autonomous Vehicles for the Overtaking Maneuver". In: *IEEE Trans. Intelligent Transportation Systems* 9.3 (2008), pp. 438–450. DOI: 10.1109/TITS.2008.922880. URL: https://doi.org/10.1109/TITS.2008.922880.

[15] Daniel Ratner and Manfred K. Warmuth. "Finding a Shortest Solution for the N × N Extension of the 15-PUZZLE Is Intractable". In: *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, August 11-15, 1986. Volume 1: Science.* Ed. by Tom Kehler. Morgan Kaufmann, 1986, pp. 168–172. URL: http://www.aaai.org/Library/AAAI/1986/aaai86-027.php.

[16] Daniel Ratner and Manfred K. Warmuth. "NxN Puzzle and Related Relocation Problem". In: *J. Symb. Comput.* 10.2 (1990), pp. 111–138. DOI: 10.1016/S0747-7171(08)80001-6. URL: http://dx.doi.org/10.1016/S0747-7171(08)80001-6.

[17] R. Schubert, K. Schulze, and G. Wanielik. "Situation Assessment for Automatic Lane-Change Maneuvers". In: *IEEE Transactions on Intelligent Transportation Systems* 11.3 (Sept. 2010), pp. 607–616. ISSN: 1524-9050. DOI: 10.1109/TITS.2010.2049353.

[18] H. S. J. Tsao, R. W. Hall, and S. E. Shladover. "Design options for operating automated highway systems". In: *Vehicle Navigation and Information Systems Conference, 1993., Proceedings of the IEEE-IEE.* Oct. 1993, pp. 494–500. DOI: 10.1109/VNIS.1993.585680.

[19] F. Visintainer, L. Altomare, A. Toffetti, A. Kovacs, and A. Amditis. "Towards Manoeuver Negotiation: AutoNet2030 Project from a Car Maker Perspective". In: *Transportation Research Procedia* 14 (2016). Transport Research Arena TRA2016, pp. 2237 –2244. ISSN: 2352-1465. DOI: http://dx.doi.org/10.1016/j.trpro.2016.05.239. URL: http://www.sciencedirect.com/science/article/pii/S2352146516302459.

[20] Meng Wang, Serge P Hoogendoorn, Winnie Daamen, Bart van Arem, and Riender Happee. "Optimal Lane Change Times and Accelerations of Autonomous and Connected Vehicles". In: *Transportation Research Board 95th Annual Meeting.* 16-2914. 2016.

[21] Richard M Wilson. "Graph puzzles, homotopy, and the alternating group". In: *Journal of Combinatorial Theory, Series B* 16.1 (1974), pp. 86–96.

[22] Tay Wilson and W Best. "Driving strategies in overtaking". In: *Accident Analysis & Prevention* 14.3 (1982), pp. 179–185.

Discussion and Conclusion

6 Discussion and Conclusion

We discus several different topics in distributed and centralised algorithms. This includes TDMA algorithms for wireless sensor networks, algorithms for distributed storage and an algorithm that addresses lane changes on highways.

TDMA algorithms

Within the first two papers we look into wireless networks. The first paper considers fault-tolerant systems that have basic radio and clock settings without access to external references for collision detection, time or position, and yet require constant communication delay. We study collision-free TDMA algorithms that have uniform frame size and uniform timeslots and require convergence to a data packet schedule that does not change. Our algorithm considers the timeslot allocation aspects of the studied problem, together with transmission timing aspects. Interestingly, we show that the existence of the problem's solution depends on convergence criteria that include the ratio, τ/δ, between the frame size and the node degree. We establish that $\tau/\delta \geq 2$ as a general convergence criterion, and demonstrate the existence of collision-free TDMA algorithms for which $\tau/\delta \geq 4$. This improves previous results [1], that require $\tau/(\Delta+1) \geq 2$, where Δ is the distance-2 neighbourhood. Unfortunately, our result implies that, for our systems settings, there is no distributed mechanism for asserting the convergence criteria within a constant time. For distributed systems that do *not* require constant communication delay, we propose to explore such criteria assertion mechanisms as future work.

Namely, an extended version of the proposed algorithm could be based on discovering the entire distance-2 neighbourhood by letting the control packet exchange deal with ratio of $\tau < \max(2\delta, \chi_2)$, where δ is a bound on the node degree, and χ_2 is the chromatic number for distance-2 vertex colouring. Here, each connected component needs to independently decide on a random period during which no active node transmits, and thus eventually all connected component share a period in which no active node transmits. The passive nodes on hand can use well-known backoff techniques that multiplicatively decrease the control packet rate, and thus eventually transmit during a period in which no active node transmits.

In the second paper we look into implementations of TDMA algorithms on sensor network hardware, i.e., the TelosB mote in the Indriya testbed [3]. Although, wireless sensor network applications try to save power, which contradicts our always online TDMA algorithm, we believe that our experimental results are valuable. Controlling robotic, or vehicular, systems, require real-time measurements of the environment which can be implemented using the bandwidth guarantee of a TDMA schedule. We provide two implementations. One is based on the first paper and one that is tailored to the give

hardware. While the first implementation differs only slightly from the proposed theoretical algorithm, e.g., wrapping around the maximal clock value is not implemented, the second implementation gives up self-stabilisation completely. Of course, even the first implementation is not self-stabilising any more, due to the lack of self-stabilising hardware and operating systems. The second implementation explicitly ignores weak communication links. If the link is bad due to collisions, or due to background noise is indistinguishable and therefore it is not able to detect all collisions. Since, nodes back off regularly when the background noise chances and therefore collision might be resolved, the system is most of the time in a surprisingly good state.

Our experimental results motivates to look into an upper bound on the expected convergence time for the theoretical TDMA algorithm, which is presented in the first paper, as well as the DecTDMA algorithm, which is presented in the second paper. There are two possible models in which such a bound can be discussed. One would be the communication graph which is presented in the first paper. Here the crucial task is to show the convergence of the clock synchronisation. As pointed out in the first paper, the number of different clock values is monotonically decreasing, but the clock synchronisation must work against clocks that wrap around. For example, assume there is currently a maximum value which is held by all, but one node. If the last node does not jump to this maximum value before the nodes with maximum wrap around, then the nodes which had the maximum value before the wrapping around might jump to the clock value of the single node, which is now the new maximum. Will all nodes converge before the clocks wrap around again? A good bound would take the network topology into account. A second model would represent more realistic communication channels, based either on the SNR, or the AWGN model. These models are more precisely modelling the simulation and testbed results in the second paper. They are essentially allowing to model packet loss based on noise that change over time and is either random background noise, or simultaneous transmissions. Such models allow edges in the communication graph not only to have packet success probabilities of 0 and 1, but also probabilities in between. This results in new challenges which we tackle with the link quality estimation of DecTDMA in the second paper, but which are not yet theoretical discussed for our solutions.

Shared memory emulation

The third paper presents a way of adding privacy to an existing multi-reader multi-writer shared memory algorithm. By extending the CAS algorithm from Cadambe et. al [2] that is based on the I/O Automata model of Lynch [5]. We use the Reed-Solomon code [12] with is widely used in storage solutions, like optical disks [10], or hard drive RAID systems [11] and which can be used [6] for secret sharing [13].

Lane changing

The fourth paper presents tight bounds for sorting agents on two lanes, as well as, NP-hardness for an increasing number of lanes. We do not consider constraints on the agent

dynamics, like constraints on lateral and longitudinal acceleration given by the mass, engine power [4], tire friction [9]. Furthermore, the instability of platoons [14] need to be considered. And, finally, the measurement noise needs to be considered. In an implementation, every agent only holds an estimation of the current environment, as well as, its own position.

In a bit more macroscopic picture, one can use a bit more abstract model. For example without the target lane constraint, but with a keep left if not overtaking constraint the Korteweg-de Vries equation can be utilised [7]. Similar approaches are using cellular automata [8].

Our bounds are solely based on congestion and necessary space for navigation. The space requirements are based on the size of the vehicle compared to available road segments of this size. We assume the vehicles have all the same unit size, as well as, the same default speed and the same deceleration . Thus, this bounds also apply in realistic scenarios. But, the solution might look different when considering agent dynamics.

Bibliography

[1] C. Busch, M. Magdon-Ismail, F. Sivrikaya, and B. Yener. "Contention-free MAC protocols for asynchronous wireless sensor networks". In: *Distributed Computing* 21.1 (2008), pp. 23–42. DOI: 10.1007/s00446-007-0053-x. URL: http://dx.doi.org/10.1007/s00446-007-0053-x.

[2] Viveck R. Cadambe, Nancy A. Lynch, Muriel Médard, and Peter M. Musial. "A Coded Shared Atomic Memory Algorithm for Message Passing Architectures". In: *2014 IEEE 13th International Symposium on Network Computing and Applications, NCA 2014, Cambridge, MA, USA, 21-23 August, 2014*. IEEE Computer Society, 2014, pp. 253–260. ISBN: 978-1-4799-5392-9. DOI: 10.1109/NCA.2014.44. URL: http://dx.doi.org/10.1109/NCA.2014.44.

[3] Manjunath Doddavenkatappa, Mun Choon Chan, and Akkihebbal L. Ananda. "Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed". In: *Testbeds and Research Infrastructure. Development of Networks and Communities - 7th International ICST Conference,TridentCom 2011, Shanghai, China, April 17-19, 2011, Revised Selected Papers*. 2011, pp. 302–316. DOI: 10.1007/978-3-642-29273-6_23. URL: http://dx.doi.org/10.1007/978-3-642-29273-6_23.

[4] J. K. Hedrick, M. Tomizuka, and P. Varaiya. "Control issues in automated highway systems". In: *IEEE Control Systems* 14.6 (Dec. 1994), pp. 21–32. ISSN: 1066-033X. DOI: 10.1109/37.334412.

[5] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. ISBN: 1-55860-348-4.

[6] R. J. McEliece and D. V. Sarwate. "On Sharing Secrets and Reed-Solomon Codes". In: *Commun. ACM* 24.9 (1981), pp. 583–584. ISSN: 0001-0782. DOI: 10.1145/358746.358762. URL: http://doi.acm.org/10.1145/358746.358762.

[7] Takashi Nagatani. "Modified KdV equation for jamming transition in the continuum models of traffic". In: *Physica A: Statistical Mechanics and its Applications* 261.3 (1998), pp. 599 –607. ISSN: 0378-4371. DOI: http://dx.doi.org/10.1016/S0378-4371(98)00347-1. URL: http://www.sciencedirect.com/science/article/pii/S0378437198003471.

[8] Kai Nagel, Dietrich E. Wolf, Peter Wagner, and Patrice Simon. "Two-lane traffic rules for cellular automata: A systematic approach". In: *Phys. Rev. E* 58 (2 Aug. 1998), pp. 1425–1437. DOI: 10.1103/PhysRevE.58.1425. URL: https://link.aps.org/doi/10.1103/PhysRevE.58.1425.

[9] E. Ono, S. Hosoe, Hoang D. Tuan, and S. Doi. "Bifurcation in vehicle dynamics and robust front wheel steering control". In: *IEEE Transactions on Control Systems Technology* 6.3 (May 1998), pp. 412–420. ISSN: 1063-6536. DOI: 10.1109/87.668041.

[10] Taegeun Park. "Design of the (248,216) Reed-Solomon decoder with erasure correction for Blu-ray disc". In: *IEEE Transactions on Consumer Electronics* 51.3 (Aug. 2005), pp. 872–878. ISSN: 0098-3063. DOI: 10.1109/TCE.2005.1510497.

[11] James S Plank et al. "A tutorial on Reed-Solomon coding for fault-tolerance in RAID-like systems". In: *Softw., Pract. Exper.* 27.9 (1997), pp. 995–1012.

[12] Irving S Reed and Gustave Solomon. "Polynomial codes over certain finite fields". In: *J. Society for Industrial & Applied Math.* 8.2 (1960), pp. 300–304.

[13] Adi Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176.

[14] DVAHG Swaroop and JK Hedrick. "Constant spacing strategies for platooning in automated highway systems". In: *Journal of dynamic systems, measurement, and control* 121.3 (1999), pp. 462–470.